NAME: ONG KHOON KOK

NO. MATRICULATION: WEK020202

SUBJECT CODE: WXES 3182

PROJECT TITLE: PACKET SNIFFER & ANALYZER

SUPERVISOR: MR. NOR BADRUL ANUAR JUMA'AT

MODERATOR: MR. LING TECK CHAW

ABSTRACT

The communication of network is a complex process which cannot be seen by human beings. Even a computer that sit on a network can only knows the conversation within itself and others computer it talk to but not others computers' conversation that it didn't take part. Furthermore, the communication in network can be viewed as traffic which always triggers a lot of unsolved problems for computers to "talk". Therefore, to solve the problems we need to exactly know what really happened inside the network traffic. A variety of sniffing tools are invented for these purpose

All of these tools have the basic capability that is to capture all frames transmitted through the network without "talk" to others. Every frames transmitted reveal everything about the information how it transmit, from where it direct to, what rules (protocol) it used and what data it carried. Therefore the captured frames can then process by the programs to interpret into a meaning data that helps the user in multiple purposes. Besides the gathering of these packets will provide another means of useful information about the entire network traffic statistic.

Sniffer has the great potential to further develop for more powerful tools like network monitor, security tools or scanning tools that are intelligent to report an alert to human or automatically take the right action to prevent a certain threat for the network without intervention by humans or at least with minimum human loads.

The development of these tools has it limitations with different platforms. This is because different operating system operates differently, the sniffer program actually communicate with the kernel system of operating system and the network adapter's driver in order to turn the network card into promiscuous mode.

ACKNOWLEDGEMENT

After few months of surveying and studying, my thesis for WXES 3181 and WXES 3182 has been accomplished. Even though there may be amendment done for iteration, correction and enhancement, I have gain what I expected in further real implementation of the project. I am gratefully acknowledging peoples who give me a hand in this project.

First of all, I would like to thank Faculty of Computer Science and Information Technology (FCSIT) for providing this opportunity to me to take these meaningful subjects, Latihan Ilmiah WXES 3181 and WXES 3182. Through these courses, I have opportunity to train myself individually on IT fields especially in networking and gain experience on how to develop complex software. This is very useful to help me in my career in future.

I would like to thank my supervisor, Mr. Nor Badrul Anuar Juma'at for his guidance and taught. He had been paid his effort on supervised me on my system that I develop and help me to solve my problems. My sincere appreciation also goes to my moderator for this project, Mr. Ling Teck Chaw for his constructive criticism and valuable suggestions that had open my mind to put more efforts on the project.

Thanks also to all lecturers that had thought me in the course that related to my project either in network fields or in software engineering especially Ms. Su Moon Ting, Dr.Phang Keat Keong, Mr Ling Teck Chaw, Mrs. Norazlina Khamis and Ms. Chew Esyin. The knowledge I obtain from these courses had helped me a lot in this project.

Thanks also to all my course mates that giving me many ideas, encourage and knowledge given in accomplishing the WXES 3181 and WXES 3182 course.

CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 1 – Introduction

1.1 Overview

A packet sniffer is either a wire-tap device or a software that plugs into the network and eavesdrops on the network traffic. But in this project what I will developed is a software rather that hardware. When a computer is running a packet sniffer, its network interface is in promiscuous mode, which means that it listens for all traffic and it has the ability to receive all of the packets reaching the interface. Sniffer is invisible, passive and serves only as listener without invoke any network traffic load or communicate with other host.

Sniffer consists of capture driver, buffer, real-time analysis, decoder and the packet editing or transmission. Capture driver is the most important part of a packet sniffer because it captures the network traffic from the wire, filters it for the particular traffic one wants. Buffer is then used to store the frame after being captured from the network. There are a couple captures modes: capture until the buffer fills up, or use the buffer as a circular buffer where the newest data replaces the oldest data. The packet editing is optional features which allowed user to edit their own network packets and transmit them onto the network.

In this project, I will using Winpcap which is an architecture and also a frame work that adds to operating system of windows family the ability to capture data through the network adapter. Winpcap provide the capture driver for which works at the kernel level of the operating system.

After the packets is captured, it is analyzed in real-time in the user-level of the architecture where the structure of the packets is reveal according to the protocol of the

packet at different layer of TCP/IP protocol. The analyzed data is then displayed in a human readable format. The summarized of the packets captured will be able to present us the statistic of IP traffic or protocols.

1.2 Problem Statement

Networks are more the key to business operations than ever before. If the network is not functioning properly then business can be impacted significantly, resulting in loss of revenue as well as end user productivity. Therefore there was a need for network professionals to cover every aspect of network management and network security.

The communication of network is a complex process. All the communications on a network are origin at a source and travel to a destination. The information that is sent on the network is referred as data packets. The data packets are transforming to electronic signal before they are travel through the physical cable. We can confirm the transmission of the information is successful only if the data packets finally receive by the receiver and they are readable (without loss of data). This does not provide sufficient information for troubleshooting if any failures of transmission have occurred.

In certain company, there are rules and policies to control the usage of the Internet or network where certain application or sites are forbidden. These examples are chatting programs and site that providing downloading of movies of music files like Napster, Kazaa etc. There is no way to make sure the employees are doing the right things in the right time by using the Internet resource unless we supervisee them without their knowledge.

The same problem arise in colleague or schools where students access to internet for entertainments, phonograph, chatting, downloading infringement files (movies or music) or even playing network games instead of educational purpose. The monitoring of what they done online is not limited here but it is needed if the examination is conducted through online. Therefore a sniffer can be used to detect any kinds of cheating and communication during the examination.

As I mention earlier, network communication is a complex process, therefore developing a network program is quite troublesome as we can't see what exactly the packet looks like and also we can't see if the packets was really sent to or received from other host. This makes the process of debugging difficult and is always time consuming.

Learning networks will be more interesting if we can practically see the real packets that pass through the network in real time. By analyzing the real packets, students can more understand the mechanism of the network protocols which make studying more effective.

Network security is a critical issue that must be considered whenever a new network is setup. Networks are vulnerable to security threats like intrusion, inceptions virus attack, eavesdropping and denial-of-service. Physical security is not enough as there could be internal attackers the one who launch the attacks. Hence there was a need for us to monitor the traffic in order to detect any threats occurs.

1.3 Project Objective

Many tools are designed to help administrator in troubleshooting and monitoring the networks. One of the simple and useful tools is packet sniffer and analyzer. The tool

is able to capture the packets in the LAN without increasing the traffic of the network. The packets captured will be useful information for network administrator to analyze the network traffic. Moreover, a packet sniffer performs fault analysis in order to discover problems in the network, such as why computer A cannot talk to computer B and performance analysis to discover network bottlenecks.

A security administrator uses multiple sniffers, as a detection system by placing them strategically throughout the network. A packet sniffer converts the data to human readable format so that people can read the traffic. It can help administrators perform network intrusion detection in order to discover intruders and network traffic logging in order to create logs that eavesdroppers cannot break into and erase.

Packet sniffer implies that a intruder can use this technique to automatic sift of clear-text passwords and usernames from the network, in order to break into systems, or any other useful information such as credit card numbers. From this moment, packet sniffing becomes a threat that has to be taken into account. It is a useful tool for eavesdroppers or hackers.

In company, packet sniffer helps the high level of management solved one of the most critical problems for them. That is ensure company network policies are followed by employees. With the tool, it means that it reveal everything about who is doing what and how the company network resource is used. Besides they can also get a real time picture of top bandwidth usage nodes and targets.

For parents, the sniffer is a good tool enough to supervise what their kids doing online. You may also need to detect whether your privacy is sending out by some "Adware" or "Spyware". Packet sniffer is able to reveal the connection that establish

between two hosts, therefore it can be used to detect any hosts talk to each others when examination is conducted through the online system.

If you are an associate C++/Java/ASP/JSP/PHP/SOAP network Programmer, packet sniffer with its analyzing ability can help to fix problems immediately in your newly deployed applications. It is a handy tool to assist you in debugging and testing by examining every packets and messages. For example if you want to know the destination and source port number for the packets sent by you newly created application, you can using sniffer to capture the packet and it will help you analyzed it to shows the port number. Students after learning the variety mechanism of the protocol can use the sniffer to discover and enhance what they learn previously by the studying the packets analyzed.

The use of packet sniffer is not ended here. It can be further developed to more advanced online detection software. Carnivore is a controversial program which make used of sniffing technology was developed by the U.S. Federal Bureau of Investigation (FBI) to give the agency access to the online/e-mail activities of suspected criminals.

1.4 project scope and limitation

The robustness of sniffer will depend on how far it sniffs with minimum usage of CPU processing, memory loaded, customized features available, and how far it able to analyze the packets captured to provide useful information.

A simple sniffer can only working with LAN device like Hub or Repeater but not with switch or router. But in real world, most network are constructed using switch or router rather than hub as it will reduces collision domain and broadcast domain. In

switch, only broadcast and multicast packet can be received by sniffer unless the switch support "port mirroring"or "port spanning" which is a feature that allows us to configure the switch to redirect the traffic that occurs on port to a designated monitoring port on the switch.

There are many protocols in every layer of the TCP/IP model a packet may contain, for example IP, TCP, UDP, SMTP, HTTP, TELNET, FTP and etc. It is timing consuming to develop an application that could analyze all the protocols, thus in this project I will only analyzed the packets at data-link layer, network layer and transport layer and focus on the most important protocols like Ethernet, IP, TCP, UDP and ARP.

1.5 Project Schedule

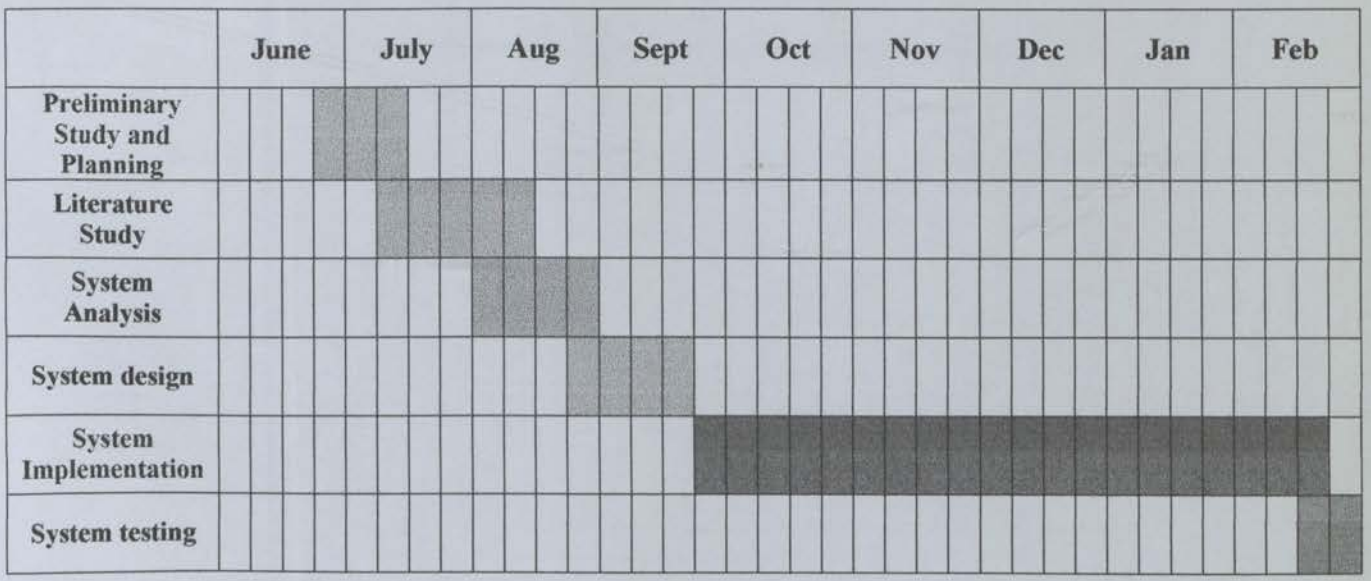


Figure 1.5.1: Project Schedule

Chapter 2

Literature Review

Chapter 2 - Review of Literature

Literature review is a critical look at the existing research that is useful and significant to the work that is carried out.

2.1 Analysis Studies

In this stage, finding, summarize, and analyze of the existing system will be done. This is to ensure the understanding of the software that will be developed and to get the requirement of the software.

2.1.1 Case Study 1 – IP Promiscuous Sniffer [1]

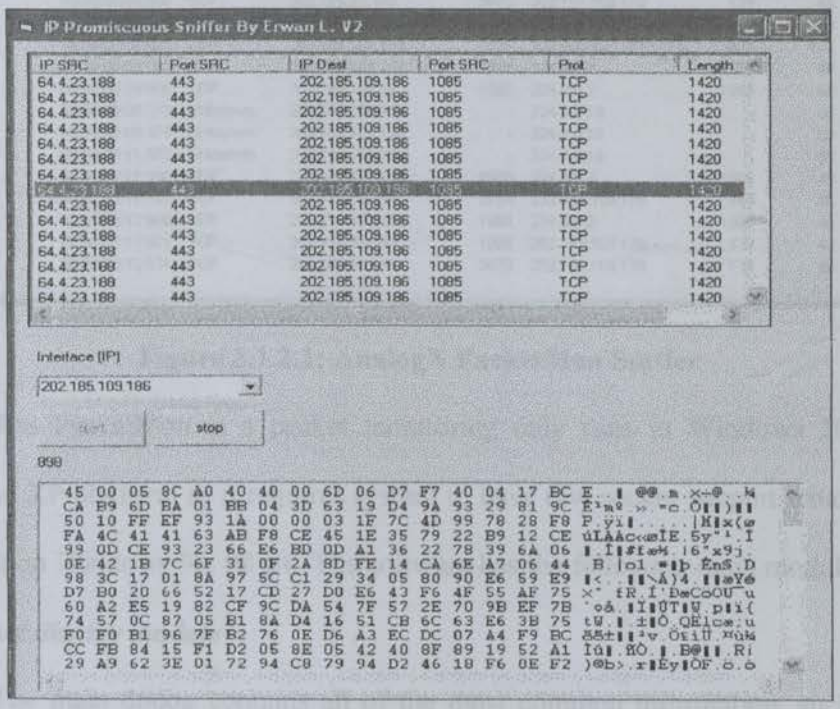


Figure 2.1.1.1: IP Promiscuous Sniffer

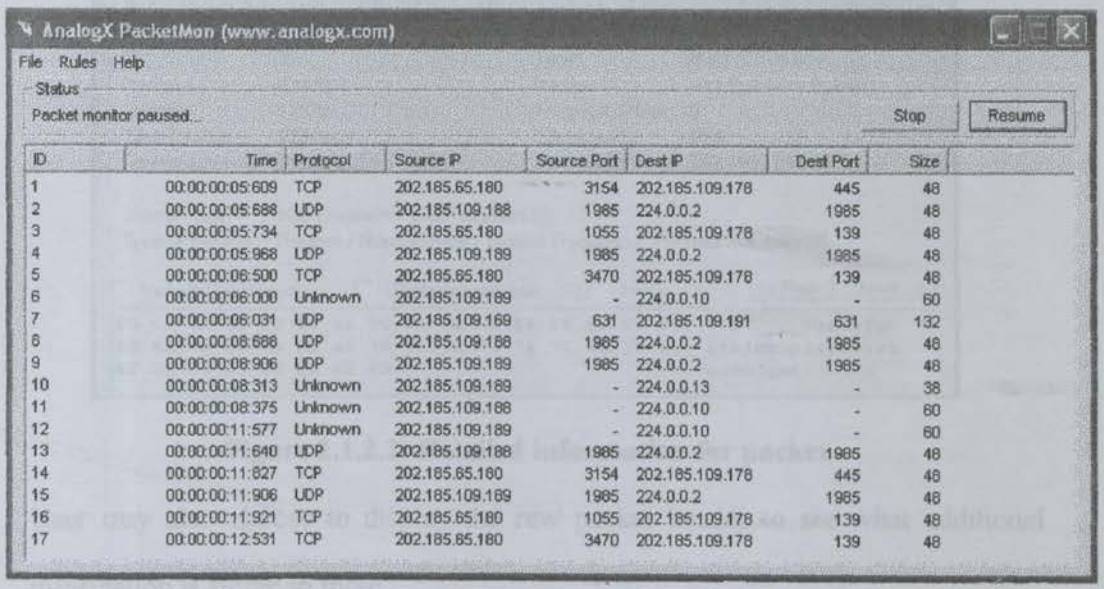
The IP Promiscuous Sniffer is a simple packet sniffer written in visual basic and it works under windows 2000 and windows XP with no driver needed for it to run.

This program has one main module:

- ***packet display module***

The program lists the packets captured in a table with some information on IP header (IP address, protocol) and TCP header (port number). A click on the packet on the packet will result the display of the content of the raw packet.

2.1.2 Case Study 2 – AnologX PacketMon [2]



ID	Time	Protocol	Source IP	Source Port	Dest IP	Dest Port	Size
1	00:00:00:05:609	TCP	202.185.65.180	3154	202.185.109.178	445	48
2	00:00:00:05:688	UDP	202.185.109.188	1985	224.0.0.2	1985	48
3	00:00:00:05:734	TCP	202.185.65.180	1055	202.185.109.178	139	48
4	00:00:00:05:968	UDP	202.185.109.189	1985	224.0.0.2	1985	48
5	00:00:00:06:500	TCP	202.185.65.180	3470	202.185.109.178	139	48
6	00:00:00:06:000	Unknown	202.185.109.189	-	224.0.0.10	-	60
7	00:00:00:06:031	UDP	202.185.109.169	631	202.185.109.191	631	132
8	00:00:00:08:688	UDP	202.185.109.188	1985	224.0.0.2	1985	48
9	00:00:00:08:906	UDP	202.185.109.189	1985	224.0.0.2	1985	48
10	00:00:00:08:313	Unknown	202.185.109.189	-	224.0.0.13	-	38
11	00:00:00:08:375	Unknown	202.185.109.188	-	224.0.0.10	-	60
12	00:00:00:11:577	Unknown	202.185.109.189	-	224.0.0.10	-	60
13	00:00:00:11:640	UDP	202.185.109.188	1985	224.0.0.2	1985	48
14	00:00:00:11:827	TCP	202.185.65.180	3154	202.185.109.178	445	48
15	00:00:00:11:906	UDP	202.185.109.189	1985	224.0.0.2	1985	48
16	00:00:00:11:921	TCP	202.185.65.180	1055	202.185.109.178	139	48
17	00:00:00:12:531	TCP	202.185.65.180	3470	202.185.109.178	139	48

Figure 2.1.2.1: AnalogX PacketMon Sniffer

The PacketMon is a packet monitoring only runs in Windows 2000 and Windows XP platform as it requires system to have raw socket support which is not available on Windows 95, 98 etc. The software has the following main modules:

- ***packet display module***

The main dialog contains all of the most common information, such as the source IP address, destination IP address, protocol, etc. User can get a more detailed view of the specific packet, simply double-click on the list, and the detailed view will be opened.

As shown in the figure below the top portion of the dialog represents the decoded IP packet header while the bottom portion contains the actual contents of the packet. The next and prev buttons allow user to move forward and backward to the adjacent packets. If the 'Trace' option enabled, it will only step forward to packets which share the same source and destination IP's and ports, to help us view a specific connection.

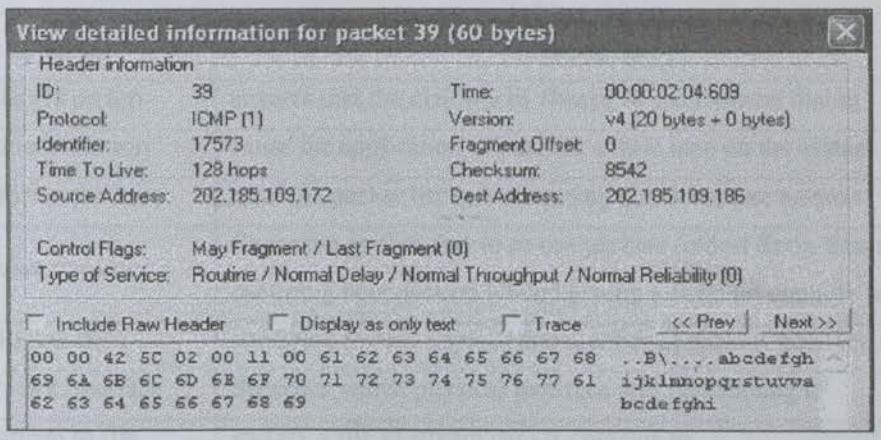


Figure 2.1.2.2: Detailed information for packet

User may also choose to display the raw packet header to see what additional information is stored in there.

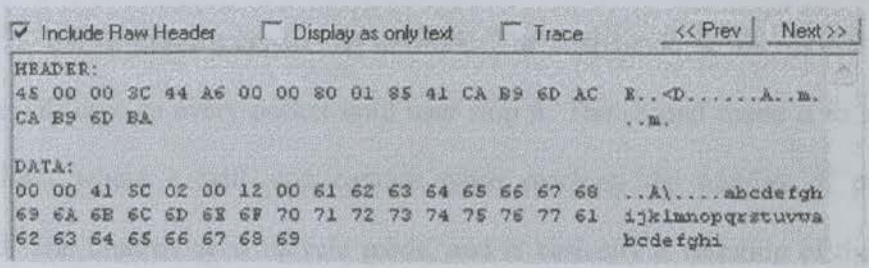


Figure 2.1.2.3: Include raw header function

- ***configuration module***

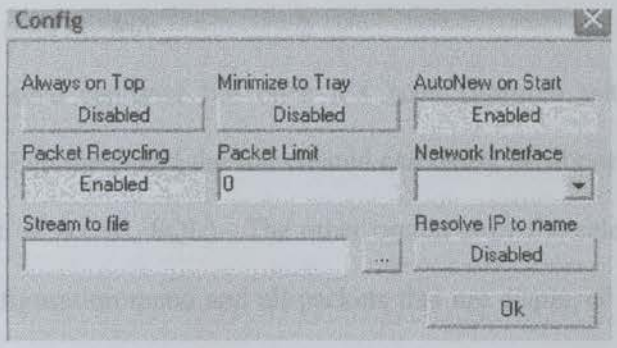


Figure 2.1.2.4: config menu

function	description
Always on top	ensures that the dialog will always be the topmost dialog
Minimize to tray	cause the application to become a little icon on the system tray
AutoNew on start	reset the packet list before starting a new capture session
Packet recycling	Cause the sniffer start to re-use packets (oldest first), thus still capturing new packets while keeping a constant memory usage.
Packet limit	configure to only capture that number of packets and then stop
Network interface	Specifies which physical interface will be listening to
Stream to file	specify a file to stream packet contents to
Resolved IP to name	causes the program to resolve as many IP addresses as it can

Table 2.1.2.1: configuration menu

Note that PacketMon has three different modes it can run depending on the amount of traffic moving across the network. The first mode is the normal mode where it will capture every packet until user stop it. The second mode is to set the packet limit where it will stop capture when reaching the number of packets captured. The final mode is recycle mode, and is basically a variation of the limit mode. When a limit is set, and recycle is turned on, the sniffer will start to re-use packets (oldest first), thus still capturing new packets while keeping a constant memory usage. This mode is more for people who are interested in monitoring a large amount of information and are using it in conjunction with the stream to file option.

➢ ***exporting data module***

User can save captured file once it's been captured with the 'Save As' function from the File menu and the file created can easily be imported into a variety of programs (such as access, SQL). The other option is by the select the 'Stream to file' from the configuration menu and all packets that are displayed will be stored to the file specified.

➢ ***filter module***

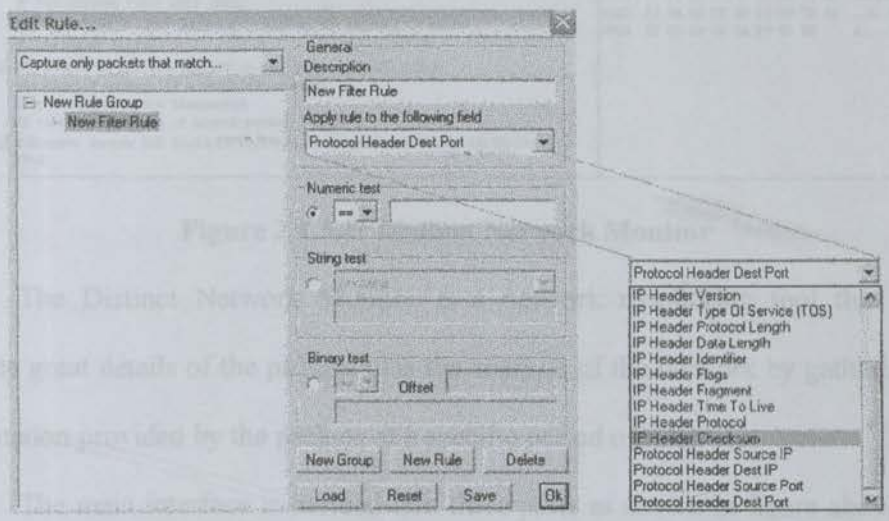


Figure 2.1.2.5: Edit Rule function

The module has three options either set the rule to capture only that match or rule that didn't match or disable the filter function. User may begin setting the new rules by create the new group which contain many rules. For each rules, there will be many fields in the IP header for user choose to apply. For each rule there will be three terms to create the rules. There are Numeric test, string test and binary test.

2.1.3 Case Study 3- Distinct Network Monitor [3]

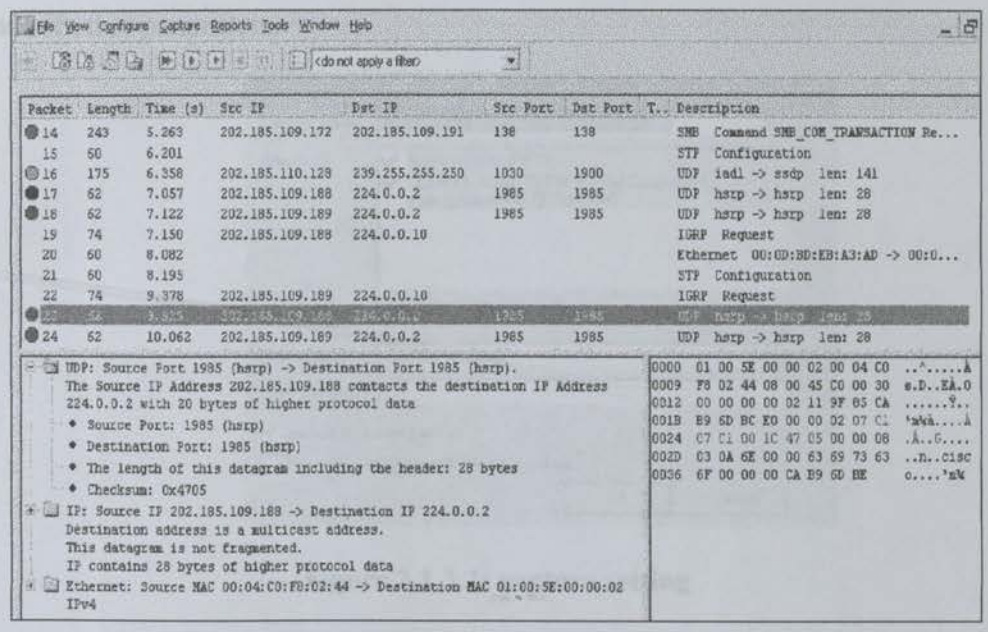


Figure 2.1.3.1: Distinct Network Monitor

The Distinct Network Monitor is a network monitoring tool that could provide great details of the packets plus the analysis of the network by gathering the information provided by the packets at a specific period of time.

The main interface is divided into three parts as shown in figure above. The first is the listing of the packet captured in a table where the information like source and destination IP address length of the packets etc. Below it is a detailed view of the packet analysis at three different layers, which is transport layer (TCP), network layer (IP) and data link layer (Ethernet). The third part is the listing of the raw packet when user selects the specific packet.

The Distinct network monitor has the following main modules:

- ***capture setting module***

User can choose to start sniffing with capture of packets and statistics or capture only statistic the first time you start a capture, the Capture Settings dialog

box is displayed allowing you to choose the folder in which to save the capture files and set other parameters.

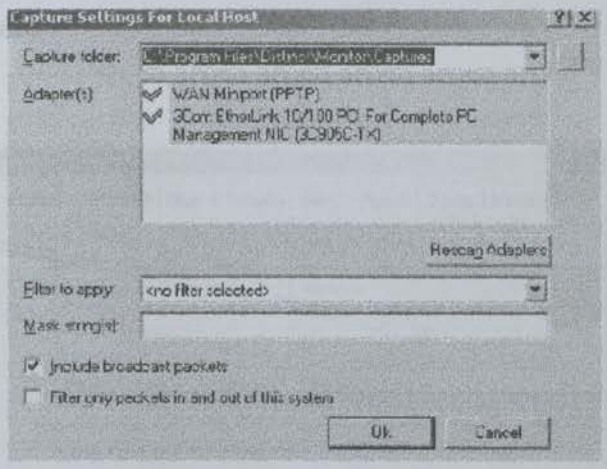


Figure 2.1.3.2: capture setting

For the first time to run Distinct Network Monitor, users need to define their capture settings for local host. A dialog box will automatically be displayed to do this. These settings can be modified later by selecting capture settings in the configure menu.

1. Enter the complete path of the folder in which you wish the capture file to be created or use the Browse button to locate it. Note that Network Monitor will remember the last capture setting saved for each folder.
2. Users may select more than one network adapter to record the network traffic. By default Network Monitor will capture the traffic from all the adapters.
3. **Filter to Apply** function enable users to save only traffic related to specific protocol.
4. User may also include or exclude broadcast packets from your capture.

5. User may select to capture only those packets that are sent and received from the system running Network Monitor, thus ignoring all other packets on the subnet

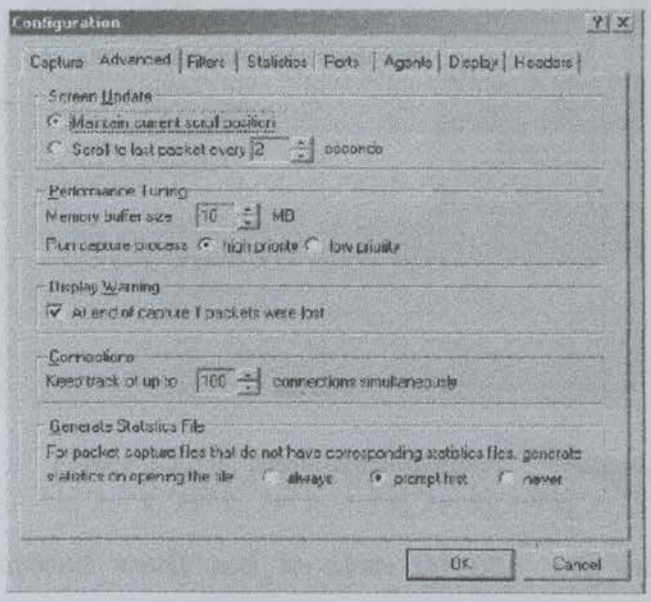


Figure 2.1.3.3: configuration

For advanced captured setting user can fine-tune the screen update, performance and display parameters. Screen update allow user to maintain the display the packet recently captured or to show only the new packet captured within a defined range of time. The performance tuning sub module allows user to specify size of memory used for the captured. The display warning is to notify user of the packet that lost while captured. The connections option configures whether to display packets from the specific connection.

➢ ***filter module***

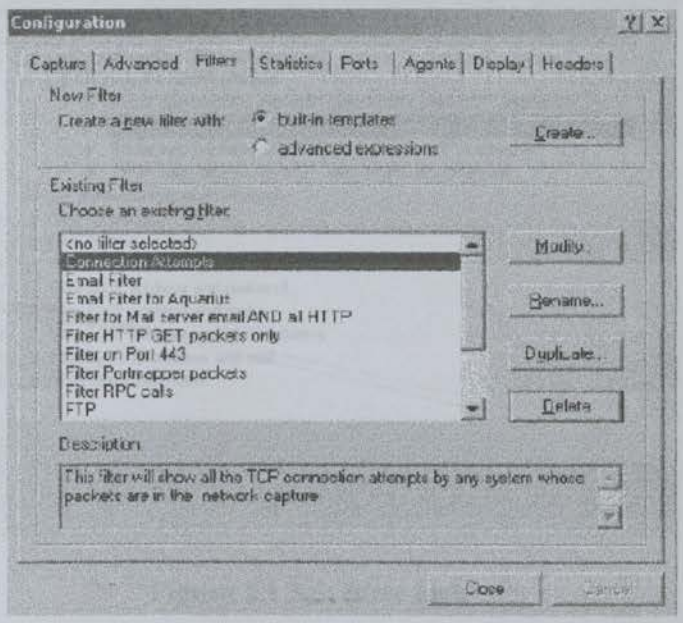


Figure 2.1.3.4: configuration-filters

The filters module enable user to create the filter with either built-in templates or advanced expressions. User may use an existing filter to or modify, rename, duplicate or delete it.

Creating the filter by template is done by selecting the terms in the tree list expanded as shown in the figure below where user can select filtering by IP address, protocol, port, or combination of them

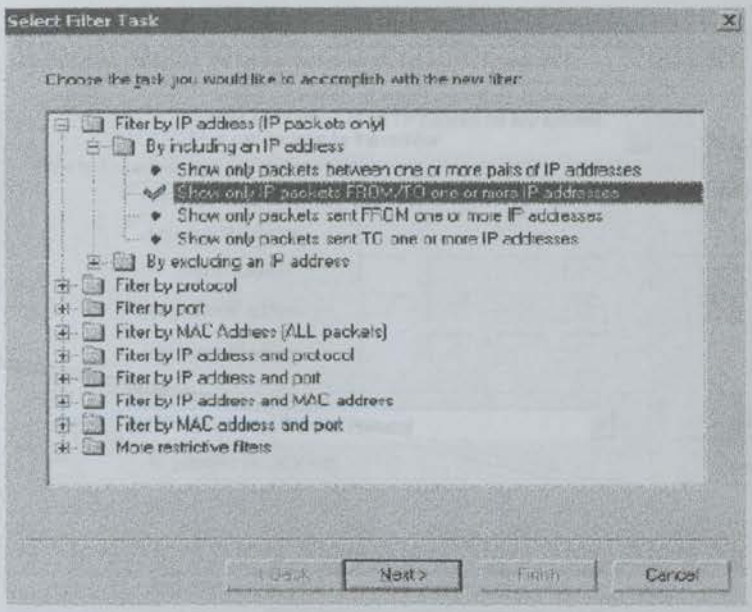


Figure 2.1.3.5: filter task

After user defined what terms he likes to filter, the program will prompt ser to enter to enter the value. For example the IP address is needed to be entered if user filters by IP protocol.

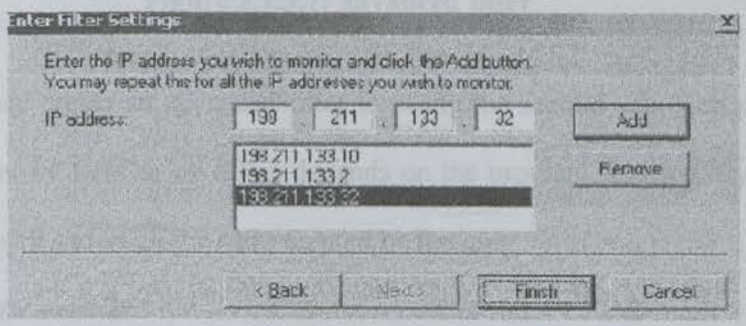


Figure 2.1.3.6: filter setting

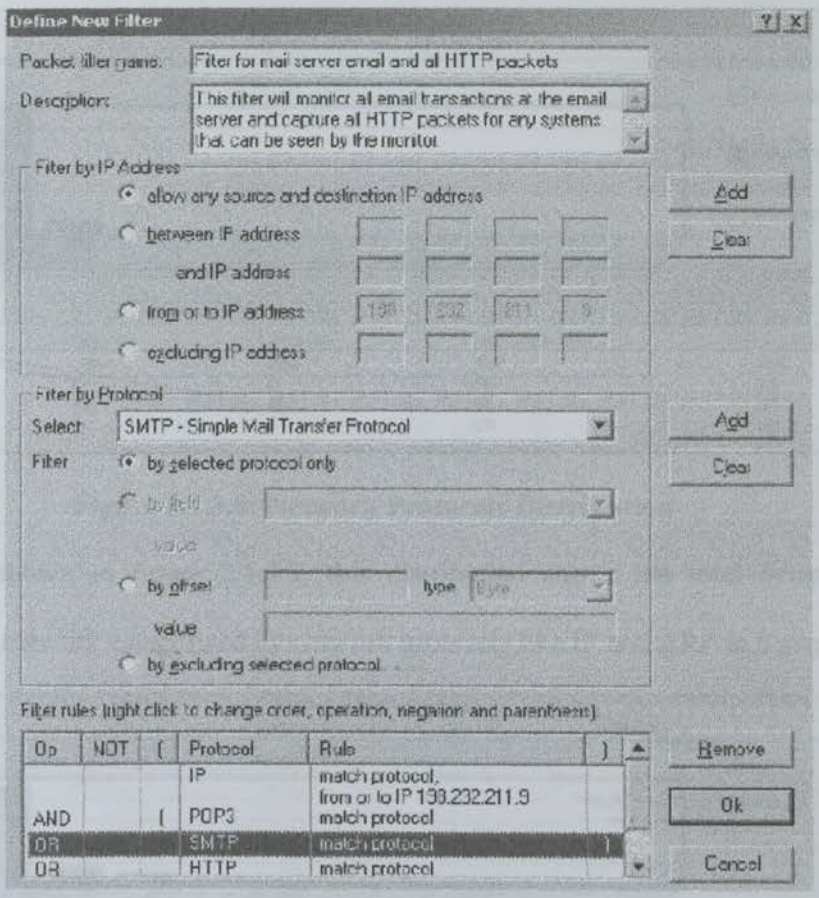
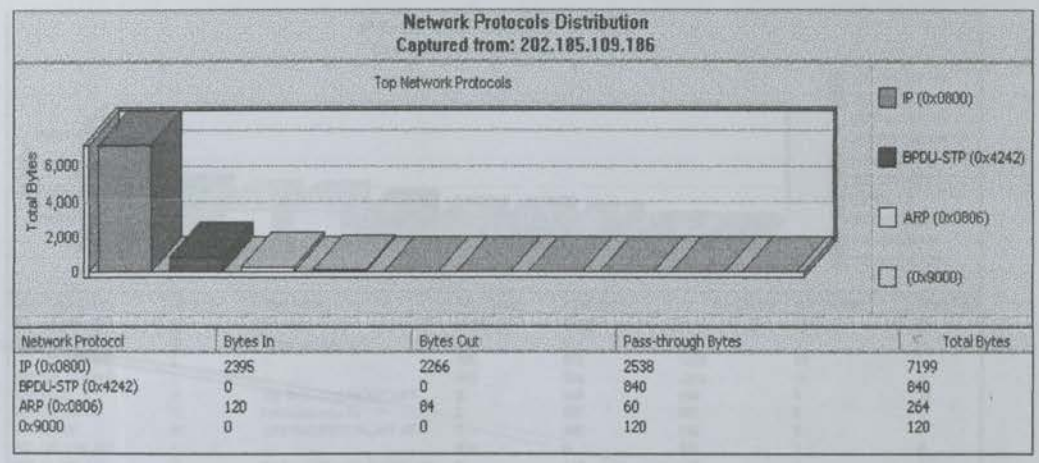


Figure 2.1.3.7: advanced filter

User may create an advanced filter by using the second approach where we can set the filter by field or by offset depends on the protocol selected. By the way the filter rules will shows all the rules created by the user.

- ***statistic module***

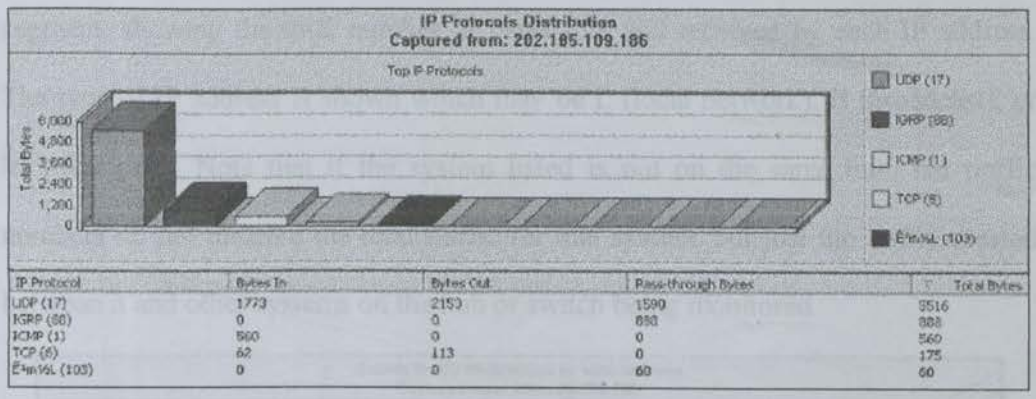
User can view the statistic of the traffic by categories it with distribution like Network Protocols Distribution, IP Protocols Distribution, IP Traffic Distribution by IP Address, Subnet Traffic Distribution by MAC address, Packet Size Distribution and Summary.



Network Protocol	Bytes In	Bytes Out	Pass-through Bytes	Total Bytes
IP (0x0800)	2395	2266	2538	7199
BPDU-STP (0x4242)	0	0	840	840
ARP (0x0806)	120	84	60	264
0x9000	0	0	120	120

Figure 2.1.3.8: Network Protocols Distribution

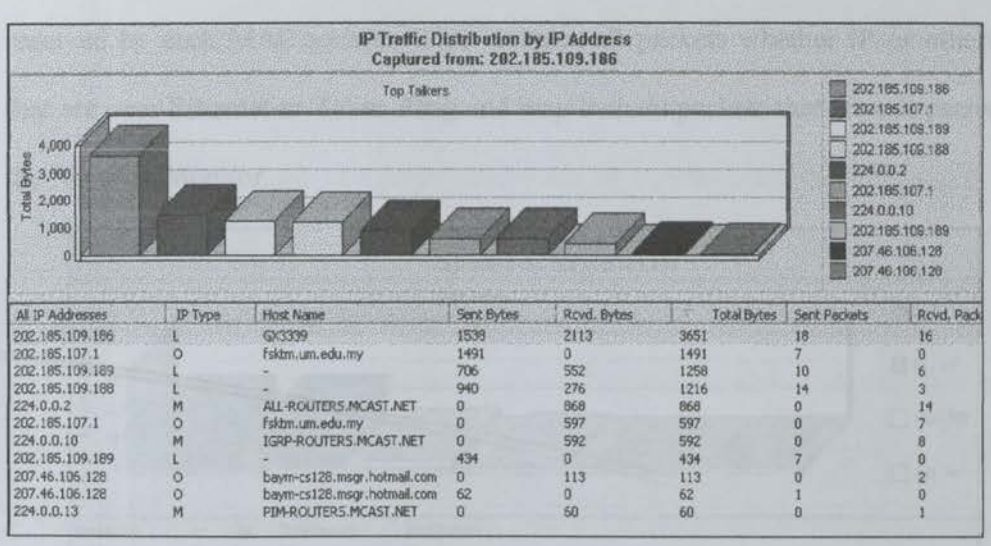
As shown in figure 2.1.3.2, this distribution shows the total bytes pass through the network categorized by network protocols like IP and ARP in a graph.



IP Protocol	Bytes In	Bytes Out	Pass-through Bytes	Total Bytes
UDP (17)	1773	2153	1590	9516
IGRP (88)	0	0	888	888
ICMP (1)	560	0	0	560
TCP (6)	62	113	0	175
[illegible] (103)	0	0	60	60

Figure 2.1.3.9: IP Protocols Distribution

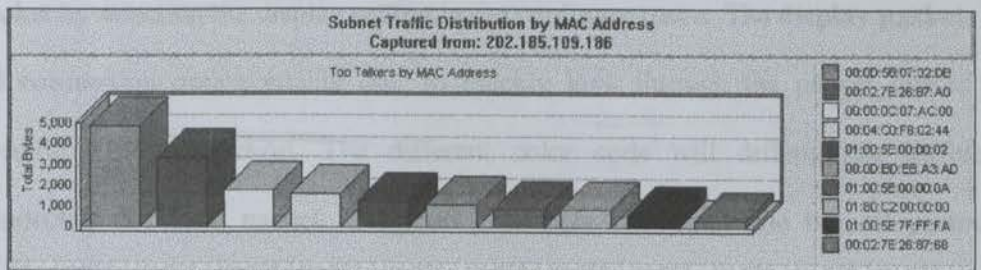
This distribution shows a list of transport protocols and the total number of bytes and packets transmitted for each one (figure 2.1.3.3).



All IP Addresses	IP Type	Host Name	Sent Bytes	Rcvd. Bytes	Total Bytes	Sent Packets	Rcvd. Pack
202.185.109.186	L	GX3339	1539	2113	3651	18	16
202.185.107.1	O	fsktm.um.edu.my	1491	0	1491	7	0
202.185.109.189	L	-	706	552	1258	10	6
202.185.109.188	L	-	940	276	1216	14	3
224.0.0.2	M	ALL-ROUTERS.MCAST.NET	0	868	868	0	14
202.185.107.1	O	fsktm.um.edu.my	0	597	597	0	7
224.0.0.10	M	IGRP-ROUTERS.MCAST.NET	0	592	592	0	8
202.185.109.189	L	-	434	0	434	7	0
207.46.106.128	O	baym-cs128.msgr.hotmail.com	0	113	113	0	2
207.46.106.128	O	baym-cs128.msgr.hotmail.com	62	0	62	1	0
224.0.0.13	M	PIM-ROUTERS.MCAST.NET	0	60	60	0	1

Figure 2.1.3.10: IP Traffic Distribution

This distribution shows the list of IP addresses that are active on the network segment, showing the total number of bytes sent and received by each IP address. The type of IP address is shown which may be L (local network), B (broadcast), or M (multicast). Note that if the system listed is not on the same hub, the traffic numbers do not indicate the total traffic for that system, but just the traffic created between it and other systems on the hub or switch being monitored.

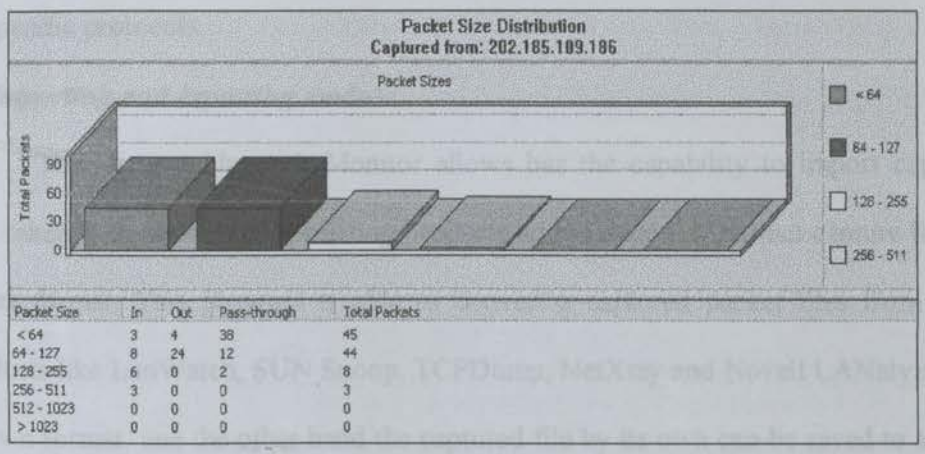


MAC Address	IP / Host Name	Sent Bytes	Rcvd. Bytes	Total Bytes	Sent Packets	Rcvd. Pac
00:0D:56:07:32:DB	GX3339	2350	2515	4865	20	20
00:02:7E:26:87:A0	202.185.109.189	2809	552	3361	24	6
00:00:0C:07:AC:00	202.185.109.189	620	1162	1782	10	14
00:04:C0:F8:02:44	202.185.109.188	1334	276	1610	20	3
01:00:5E:00:00:02	ALL-ROUTERS.MCAST.NET	0	1240	1240	0	20
00:0D:BD:EB:A3:AD		960	120	1080	16	2
01:00:5E:00:00:0A	IGRP-ROUTERS.MCAST.NET	0	888	888	0	12
01:80:C2:00:00:00		0	840	840	0	14
01:00:5E:7F:FF:FA	239.255.255.250	0	626	626	0	5
00:02:7E:26:87:68		350	0	350	2	0
FF:FF:FF:FF:FF:FF		0	144	144	0	3
01:00:5E:00:00:0D	PIM-ROUTERS.MCAST.NET	0	60	60	0	1

Figure 2.1.3.11: Subnet Traffic Distribution

This distribution shows the list of MAC addresses that are active on the network segment and the total number of packets and bytes that were sent and

received by each MAC address. This includes all packets whether IP or otherwise that are over Ethernet or Token Ring and may include packets that are not parsed by the Network Monitor.



Packet Size	In	Out	Pass-through	Total Packets
< 64	3	4	38	45
64 - 127	8	24	12	44
128 - 255	6	0	2	8
256 - 511	3	0	0	3
512 - 1023	0	0	0	0
> 1023	0	0	0	0

Figure 2.1.3.12: Packet Size Distribution

The Analysis of Packet size distribution showing the number of packets transmitted in various size ranges.

- ***packet display module***

The software enable user to customize the different way viewing of the packet by selecting the display setting in the configure menu. The **display packets of all connection** option enable user to quickly look through the packet related to specific TCP connection. The different color code will differentiate different connection for each packet. User can customize to display the highest protocol summary like FTP or the lowest protocol summary like Ethernet.

The **display of header field** can be configure to display more specific details such as TCP sequence number, windows or flags as shown in the figure below.

The **find next packet** option allows user to move from one packet with a protocol error in it to the next without having to scroll through the file. You may either select to go to the next packet with an error or the next packet with an error or warning.

The **Modify and resend** option allows user to modify and resend on the network any packet that is in a give trace file. This feature is very useful for software developers working with proprietary protocols or developing applications that rely on specific protocols.

➢ ***importing and exporting module***

The Distinct Network Monitor allows has the capability to import capture files taken with certain other sniffing products and to export a Distinct capture file to a text format. This program is able to convert a captured packet files from few products like LanWatch, SUN Snoop, TCPDump, NetXray and Novell LANalyzer to its own format. one the other hand the captured file by its own can be saved to a text file format that can be imported to other application.

2.1.4 Case Study 4- EtherDetect Packet Sniffer [4]

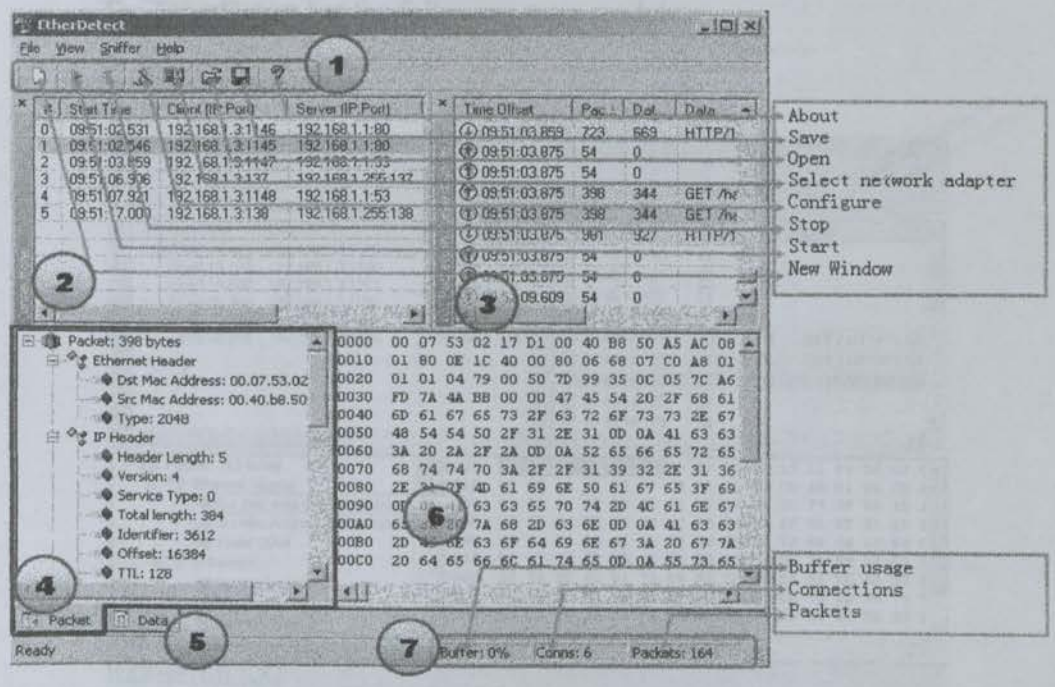


Figure 2.1.4.1: EtherDetect Packet sniffer

EtherDetect is a packet sniffer that provides a connection-oriented viewing of packets analyzing. EtherDetect is able to capture full TCP connections and UDP threads, which is essential for diagnosing programs, locating network problems, diving into network protocol, and extract application data from a variety of packets. Its powerful filter mechanism enables you setup default packet acceptance policy and customized packet-capturing filters.

Program Interface

Brief	Description
ToolBar Window	Offers the most useful commands
Connections Window	Shows a list of TCP connections and UDP threads
Packets Window	Shows a list of packets in a TCP connection or UDP thread
Protocol Tree Window	Provides protocol stack tree for a packet
Application Data Window	Provides a colorful syntax highlighting view for application data in a packet
Hex Packet Window	Provides Hex view as well as ASCII view for a packet
StatusBar Window	Shows Buffer usage, total connection number, and total packet number

Table 2.1.4.1: interface of EtherDetect

The EtherDetect has the following main modules:

- ***packet display module***

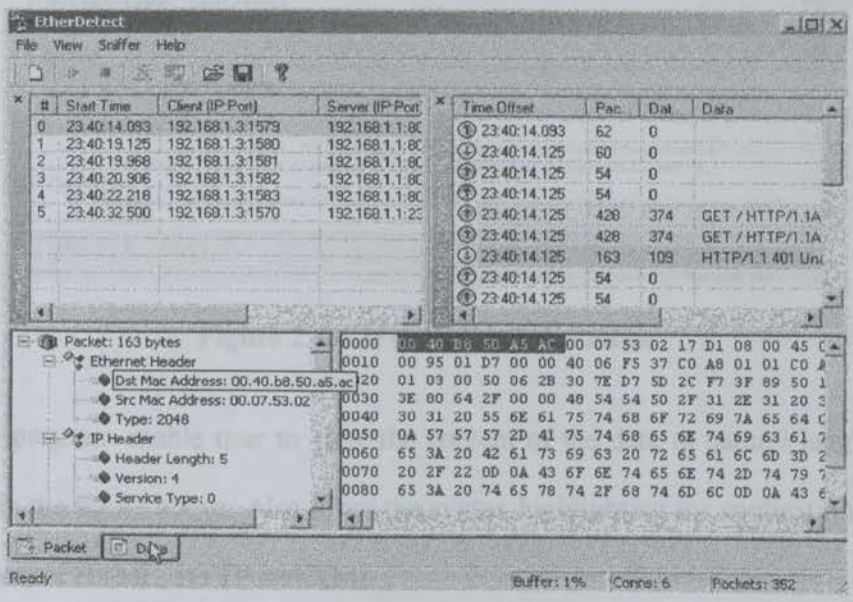


Figure 2.1.4.2: display interface

The display of the packet can be divided into four fractions which are connection windows, packet windows, protocol tree windows and hex packet windows. The display of this information is through step-by-step click by user. The connection window lists connections that occur between a client and a server. When one of the connections is selected, then the packet window will shows only all the packets for only that connection. For the same way, a click on the one of the packet in packet window will prompt the display of the information in tree windows and at the same time the display of the raw packet in hex packet windows. The information in the tree windows is obtained from the protocol header. A click on the data in the protocol tree, for example Dest Mac address will make a highlight on the associate part of raw packet in hex packet windows.

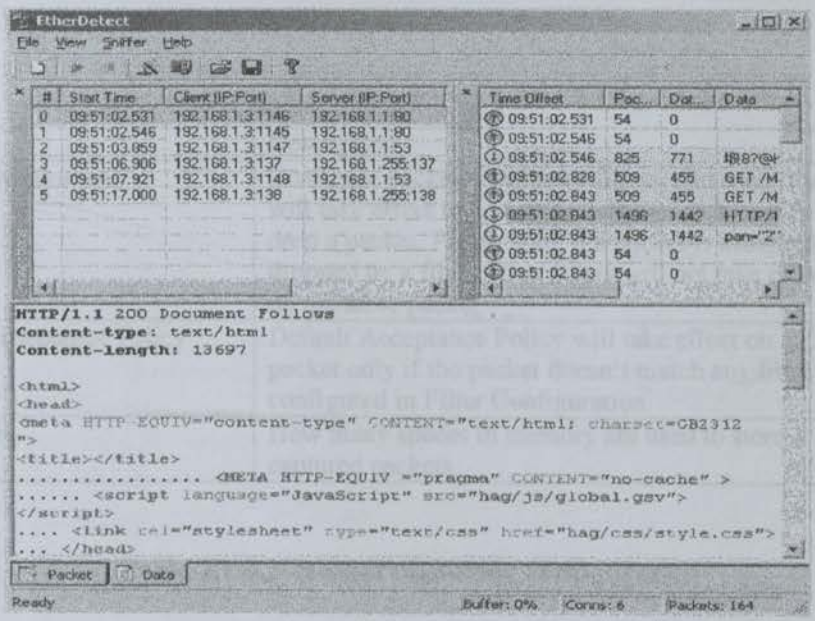


Figure 2.1.4.3: data display window

The program enable user to view the content of the data in natural language as shown in the figure above. Also it has the syntax highlighting for application data in the format of HTML, HTTP and XML.

➢ ***Filter configuration module***

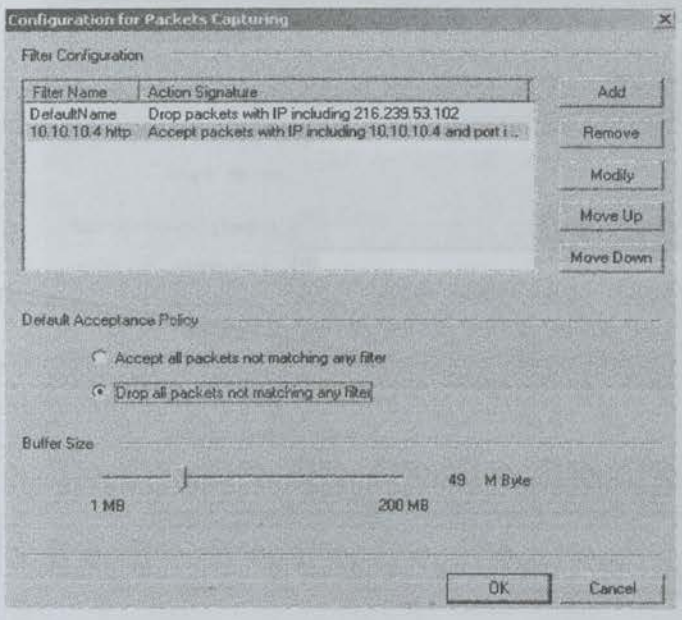


Figure 2.1.4.4: filter configuration

Brief	Description
Filter Configuration	Configure packet acceptance filters, and each filter will take effect to determine whether to accept or drop a packet. Please note, if a packet is accepted or dropped by a filter, other filters will not take effect on the same packet.
Default Acceptance Policy	Default Acceptance Policy will take effect on a packet only if the packet doesn't match any filter configured in Filter Configuration.
Buffer Size	How many spaces in memory are used to store captured packets.

Table 2.1.4.2: packet capturing configuration

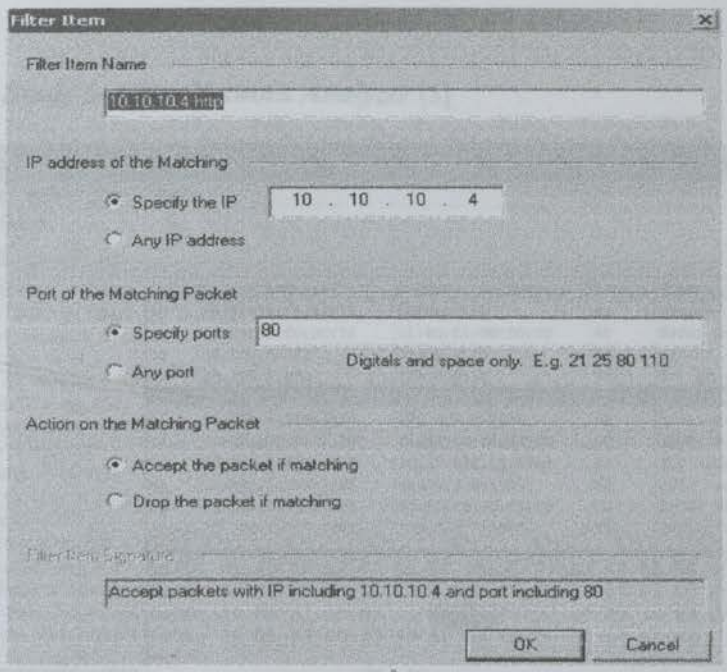


Figure 2.1.4.5: filter item

Brief	Description
Filter Item Name	The name for the filter item.
IP address of the Matching Packet	Specifies matching an IP or any IP for the filter item.
Port of the Matching Packet	Specifies matching a list of ports or any port for the filter item.
Action on the Matching Packet	Specifies whether to accept or drop the packet if it matches the IP and port condition in the filter item.
Filter Item Signature	Shows description for the filter item automatically while configuring.

Table 2.1.4.3: filter menu

- ***save and open file module***

This module enable user to save the captured packet as type *.cap. User may open the .cap file that created.

2.1.5 Case Study 5- Ultra Network Analyzer [5]

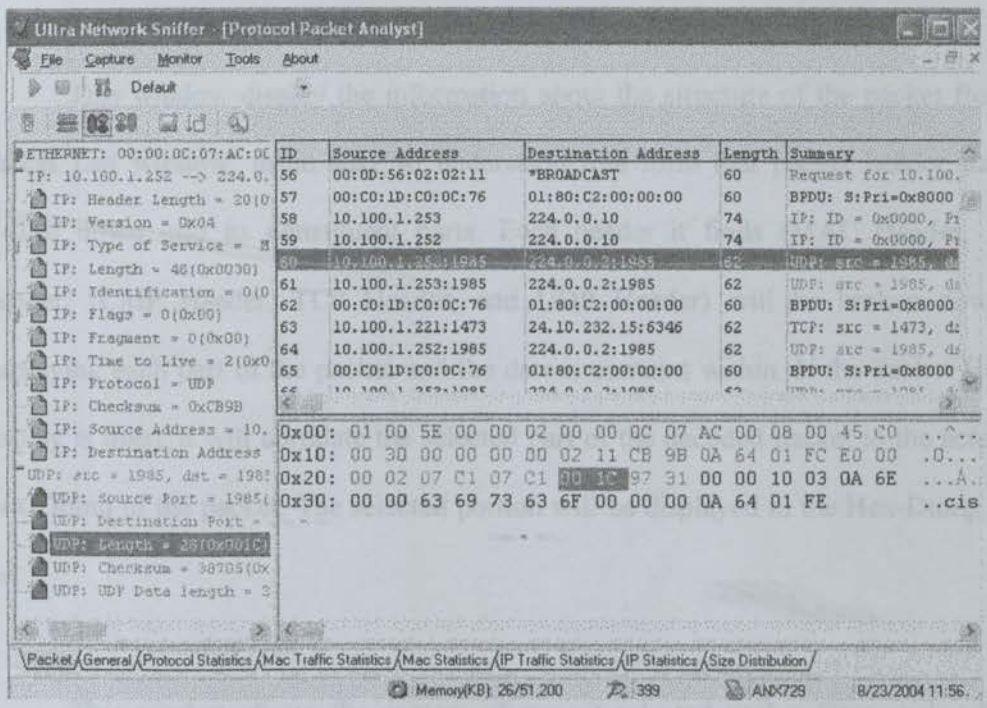


Figure 2.1.5.1: Ultra network Sniffer

Ultra Network Analyzer is a multi functional tool combination of network sniffer, packet sniffer, sockets sniffer and protocol sniffer that support all of Windows version from Windows 95 to Windows XP. It sniffs all of network packets in real-time from multi network card (Include Modem, ISDN, and ADSL) and also support capturing packet base on the application (SOCKET, TDI etc). Besides it has plug-ins for different protocols such as Ethernet, IP, TCP, UDP, PPPOE, HTTP, FTP, WINS, PPP, SMTP, POP3 etc. It has the following modules:

- ***packet display module***
  - ***packet list***

This window displays packets as they arrive from the wire. The packet display window allows you to select specific packets to be shown in the Decoder

Window. It also allows you to right click a specific packet and perform certain functions on it.

- ***packet decoder***

This window display the information about the structure of the packet from Packet List window, in an easy to understand tree form that packet header may broken down into its constituent parts. Each header it finds (MAC Header, IP Header, ICMP Header, TCP Header, and UDP Header) will be broken down, displaying each part of the packet and the data it contains within. If the Packet Hex display is open, it will correlate the selected part of the decoded packet to the actual Hex-Dump of the packet. The selected portion will be displayed in the Hex-Dump in red.

- ***packet Hex***

This window shows the content of current selected packet from packet list window. As shown in the figure, the content of the packet are colored according to the color in constituent part.

➢ ***filter module***

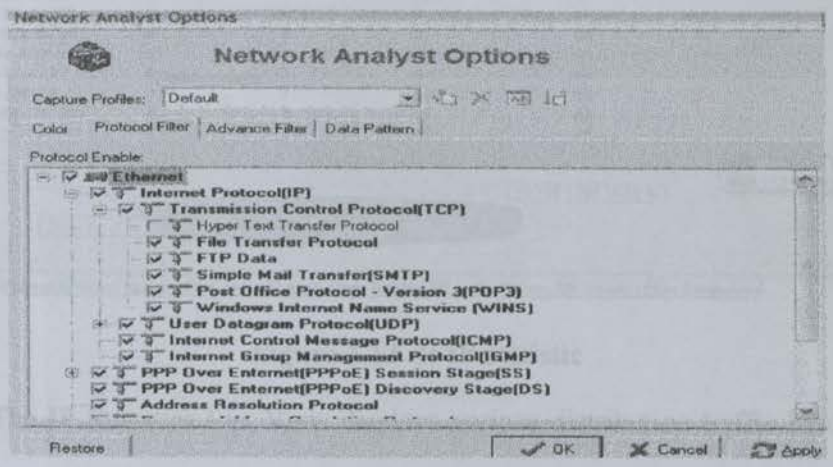


Figure 2.1.5.2: protocol filter

The protocol filter appears as a tree list with parent protocols and its child protocols. Packets that match the checked protocol in the filter will only be captured.

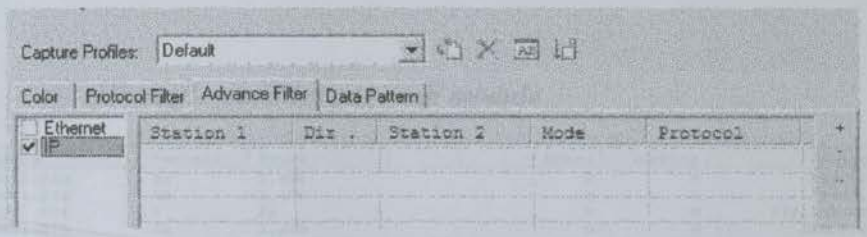


Figure 2.1.5.3: advanced filter

The advanced filter is used to set the filter in more specific way including the direction from one station to another with a specified protocol. The Mode "Include" is used for discarding all of matching packet and "Exclude" is used for only capturing all of matching packet.

- ***statistic module***
  - ***IP statistics page sub module***

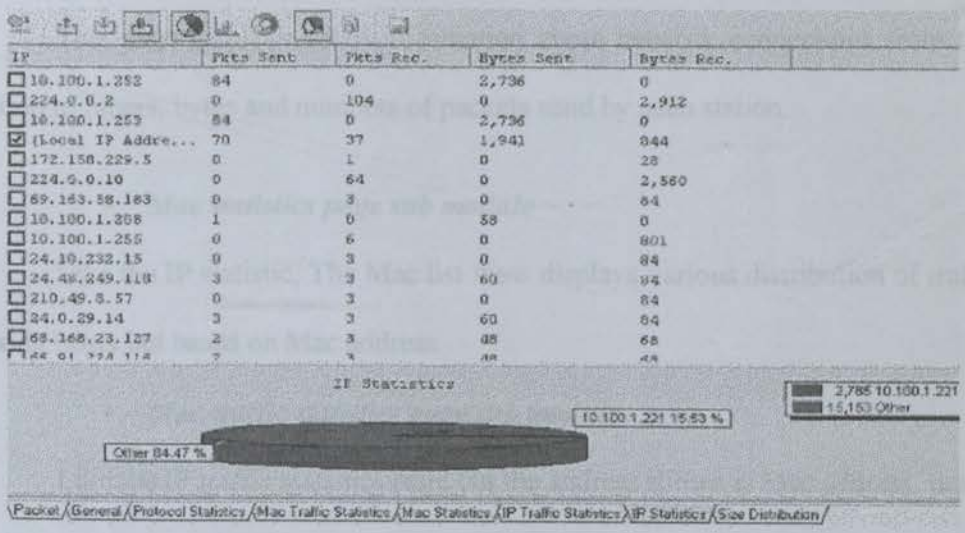
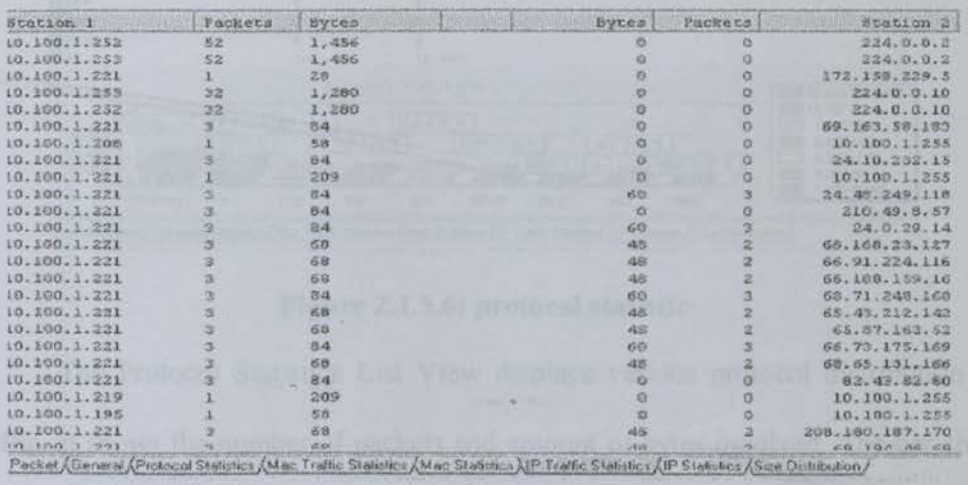


Figure 2.1.5.4: IP statistic

The IP Statistics List View displays various distribution traffic based on IP protocol including IP address, grand of sent packets, sent bytes, received packets, received bytes. User can check IP item in IP Statistics list view to chart this IP item.

IP Statistics Chart can show the number of packets or the traffic volume (bytes),also can be based on sent ,received and both.

- ***IP traffic statistics page sub module***



Station 1	Packets	Bytes		Bytes	Packets	Station 2
10.100.1.252	52	1,456		0	0	224.0.0.2
10.100.1.253	52	1,456		0	0	224.0.0.2
10.100.1.221	1	28		0	0	172.158.229.5
10.100.1.253	32	1,280		0	0	224.0.0.10
10.100.1.252	32	1,280		0	0	224.0.0.10
10.100.1.221	3	84		0	0	69.163.58.180
10.100.1.208	1	58		0	0	10.100.1.255
10.100.1.221	3	84		0	0	24.10.252.15
10.100.1.221	1	209		0	0	10.100.1.255
10.100.1.221	3	84		60	3	24.48.249.118
10.100.1.221	3	84		0	0	210.49.8.57
10.100.1.221	3	84		60	3	24.0.29.14
10.100.1.221	3	68		46	2	66.168.23.127
10.100.1.221	3	68		46	2	66.91.224.116
10.100.1.221	3	68		46	2	66.168.159.16
10.100.1.221	3	84		60	3	68.71.248.160
10.100.1.221	3	68		46	2	65.49.212.142
10.100.1.221	3	68		46	2	65.87.163.52
10.100.1.221	3	84		60	3	66.73.175.169
10.100.1.221	3	68		46	2	68.65.131.166
10.100.1.221	3	84		0	0	82.43.82.80
10.100.1.219	1	209		0	0	10.100.1.255
10.100.1.195	1	58		0	0	10.100.1.255
10.100.1.221	3	68		46	2	208.180.187.170
10.100.1.221	3	68		46	2	68.196.46.68

Packet / General / Protocol Statistics / Mac Traffic Statistics / Mac Statistics / IP Traffic Statistics / IP Statistics / Size Distribution /

Figure 2.1.5.5: IP traffic statistic

This page displays detail information about network connections from one station to others, bytes and numbers of packets send by each station.

- ***Mac statistics page sub module***

Like the IP statistic, The Mac list view displays various distribution of traffic among hosts but based on Mac address.

- ***Mac traffic statistics page sub module***

Like the IP traffic statistics page but the address shown is Mac address, not IP address.

- ***protocol statistics page sub module***

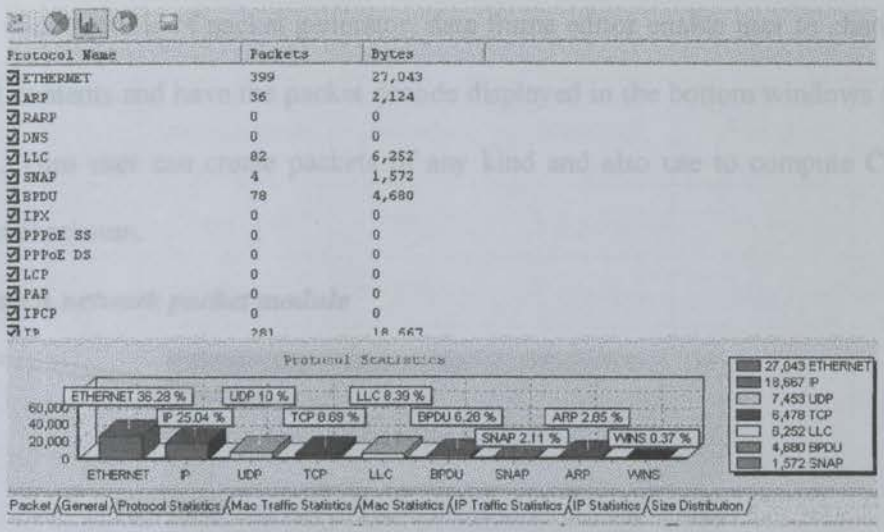


Figure 2.1.5.6: protocol statistic

The Protocol Statistics List View displays various protocol distribution of traffic. It shows the number of packets and amount of bytes involved. The bar chart shows the percentages of protocol used in the transmission.

➢ ***packet generator module***

Packet generator enable user to edit and send packets via network card.

Figure 2.1.5.7: data frame editor

Sub module of packet generator, data frame editor enable user to change the packet contents and have the packet decode displayed in the bottom windows as you edit it. Thus user can create packets of any kind and also use to compute CRC to correct checksum.

- ***search network packet module***

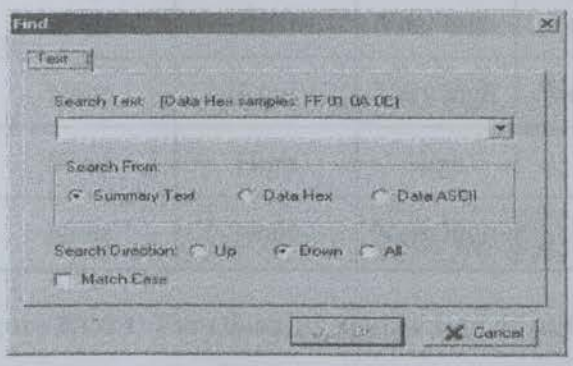


Figure 2.1.5.8: find packet

Ultra network sniffer provides the search function as in Microsoft words but the search. User may search the part of packet using different format of data like ASCII and hexadecimal.

2.1.6 Summary of Case Study

After all the case study has been analyzed their capability are compared in terms of how far the software able to configured to capture the packet in different way that help user to analyze the packet and how far the packet is analyzed to provide useful information to user. Certain software had advanced featured that others not available. The comparison of features among the five case studies is rating as Very low, low, High, and Very High. These software also assessed in term of user friendliness, reliability, manageability, stability and efficiency.

Sniffer / Features	IP promiscuous sniffer	PacketMon Network Monitor	Distinct Network Monitor	EtherDetect Packet Sniffer	Ultra Network Sniffer
Packet display	Very low	High	Very high	Very high	Very high
Packet filtering	None	High	Very high	High	Very high
Packet analyzing	None	High	Very high	Very high	High
Protocol analyzing	None	Low	Very high	Very high	High
Packet import/export	None	High	High	Low	High
Packet statistic	None	None	Very high	None	High

Figure 2.1.6.1: Functionality comparison of sniffer

Sniffer / Features	IP promiscuous sniffer	PacketMon Network Monitor	Distinct Network Monitor	EtherDetect Packet Sniffer	Ultra Network Sniffer
User friendliness	Low	Low	High	Very High	High
Reliability	Low	Low	High	High	High
Manageability	Low	High	Very high	High	Very high
Stability	Low	Very Low	High	High	High
Efficiency	Low	High	High	Very high	Very high

Figure 2.1.6.2: Non-Functionality comparison of sniffer

2.2 Technologies Review

2.2.1 LAN Technologies

The two most common network architectures are the Ethernet and Token ring. The Institute of Electrical and Electronic Engineers (IEEE) has produced a set of standards for LAN architectures. Although token ring and Ethernet were both created before the IEEE standards, the IEEE specifications for IEEE 802.3 (Ethernet) and IEEE 802.5 (token ring) now provide vendor-neutral standards for these important LAN technologies.

2.2.2 Ethernet

Ethernet and its newer sibling Fast Ethernet are the LAN technologies most commonly used today. On Ethernet networks, all computers share a common transmission medium. Ethernet uses an access method called Carrier Sense Multiple Access with Collision Detect (CSMA/CD) for determining when a computer is free to transmit data on to the access medium.

2.2.3 CSMA/CD

The Ethernet network may be used to provide shared access by a group of attached nodes to the physical medium which connects the nodes. These nodes are said to form a Collision Domain. All frames sent on the medium are physically received by all receivers, however the Medium Access Control (MAC) header contains a MAC destination address which ensures only the specified destination actually forwards the received frame (the other computers all discard the frames which are not addressed to them).

Using CSMA/CD, all computers monitor the transmission medium and wait until the line is available before transmitting. If two computers try to transmit at the same time, a collision occurs. The computers then stop, wait for a random time interval, and attempt to transmit again.

Consider a LAN with four computers each with a Network Interface Card (NIC) connected by a common Ethernet cable:

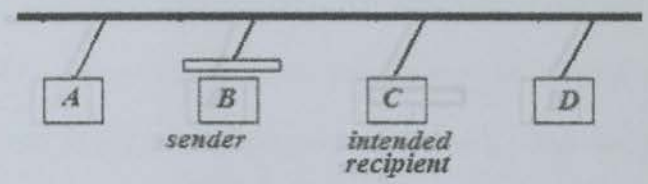


Figure 2.2.3.1: CSMA/CD

Computer B (sender) uses a NIC to send a frame to the shared medium, which has a destination address corresponding to the source address of the NIC in the computer C (intended recipient).

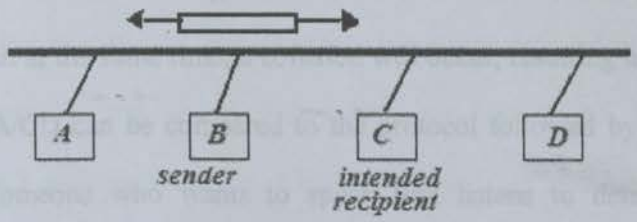


Figure 2.2.3.2: CSMA/CD

The cable propagates the signal in both directions, so that the signal (eventually) reaches the NICs in all four of the computers. Termination resistors at the ends of the cable absorb the frame energy, preventing reflection of the signal back along the cable.

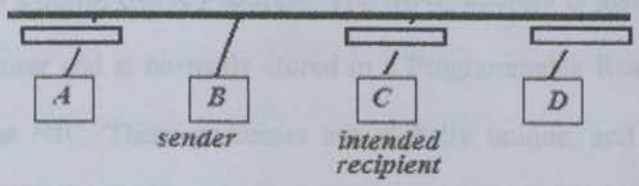


Figure 2.2.3.3: CSMA/CD

All the NICs receive the frame and each examines it to check its length and checksum. The header destination MAC address is next examined, to see if the frame should be accepted, and forwarded to the network-layer software in the computer.

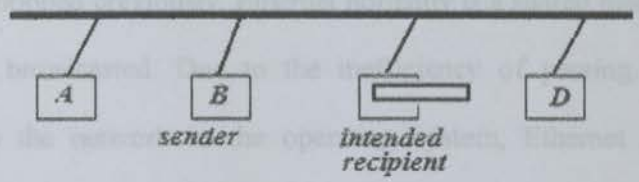


Figure 2.2.3.4: CSMA/CD

Only the NIC in the computer C recognizes the frame destination address as valid, and therefore this NIC alone forwards the contents of the frame to the network layer. The NICs in the other computers discard the unwanted frame.

The shared cable allows any NIC to send whenever it wishes, but if two NICs happen to transmit at the same time, a collision will occur, resulting in the data being corrupted. CSMA/CD can be compared to the protocol followed by a room full of polite people. Someone who wants to speak first listens to determine whether anybody else is currently speaking (this is the Carrier Sense). If two people start speaking at the same moment, both people will detect the problem, stop speaking, and wait before speaking again (this is Collision Detect).

2.2.4 Ethernet Network Interface Card

Each Ethernet NIC comes with a unique Ethernet source address called a Medium Access Control (MAC) address. The MAC address is assigned to the NIC by its manufacturer and is normally stored in a Programmable Read Only Memory (PROM)[2] on the NIC. These addresses are globally unique, and are assigned in blocks of 16 (or 8) million addresses to the Ethernet interface manufacturers, according to a flat addressing structure. This ensures that two Ethernet NICs will never have the same source address. Therefore, all NICs can be uniquely identified by its MAC address.

As I mentioned previously, Ethernet normally is a shared medium, all packets are essentially broadcasted. Due to the inefficiency of passing all the packets broadcasted on the network to the operating system, Ethernet controller chips normally implement a filter which filters out any packet that does not contain a correct destination MAC address for the NIC.

2.2.5 Principle of sniffing

The local network is usually composed of the Ethernet. On an Ethernet using IP protocol, information is sent on the cable in plain text, unless an encryption program is used. When someone sent information onto the network the only intended recipient will receive the information. The mechanism of Ethernet gives unauthorized user a chance to steal and look the data. This is because an Ethernet based network works by sending messages to all nodes on the network under the same collision and broadcast domain which only the intended node will receive the messages while others discard the messages.

Whether to receive or drop the messages is controlled by the NIC. The NIC normally don't receive all the packets come to it even it is connected to the Ethernet. Instead it filters out the packets so that only packets intended to it will be received and passes to the network layers. Filtering is done by the checking the destination MAC address of packets with the NIC's MAC address, if it is matched then that packets will be received otherwise it is discarded. Sniffing is done by setting the NIC of it's own PC to a specific mode (promiscuous mode) so that the NIC will receive whatever packet arriving to it no matter who the packets is intended.

The NIC's hardware filter can be setup to receive different type of packets. Unlike promiscuous, the normal mode of NIC's hardware filter may set to *unicast,*

broadcast and *multicast*. In unicast, all the packets having the same destination address as the hardware address of the NIC itself will be received. The broadcast packets have the destination address as FF FF FF FF FF FF and the purpose of this broadcast mode is to receive the packets which are suppose to arrive at all nodes exiting on the network. In multicast mode, all the packets which are specifically configured to arrive at some multicast group address will be received.

2.2.6 Detection of Sniffing Running [6], [7]

Even though sniffing programs are passive. It is sometimes possible to detect sniffing programs, based on the fact that the networking code of the computer running the packet sniffer generates traffic. One of detection methods is ping method. This method utilizes the fact that every machine has an IP address and an Ethernet address. The most important key here is the fact that a Network adapter running in promiscuous mode will receive any packets without checking the MAC address. Hence, one can change slightly the MAC address of the suspected machine and transmit an "ICMP Echo Request" (ping), with the right IP address but with changed MAC address. A normal PC including the suspected host shouldn't see this packet if they are running in normal mode, since each Ethernet adapter should not match the changed MAC address. If the host response, then the suspect was not running the required MAC address filter but running in promiscuous mode.

The ping method can be enhanced in a number of ways as long as any protocol that able to generates a response can be used, such as a TCP connection request or any protocol that might generate an error on the target machine. For example, bad IP header values might be used to generate an ICMP error.

2.2.7 NDIS (Network Driver Interface Specification) [8]

NDIS is a set of specifics that defines the communication between a network adapter (or its driver) and the protocol drivers (IP, IPX...). The main purpose of NDIS is to act as a wrapper that allows protocol drivers to send and receive packets onto a network (LAN or WAN) ignoring the particular adapter or the particular Win32 operating system.

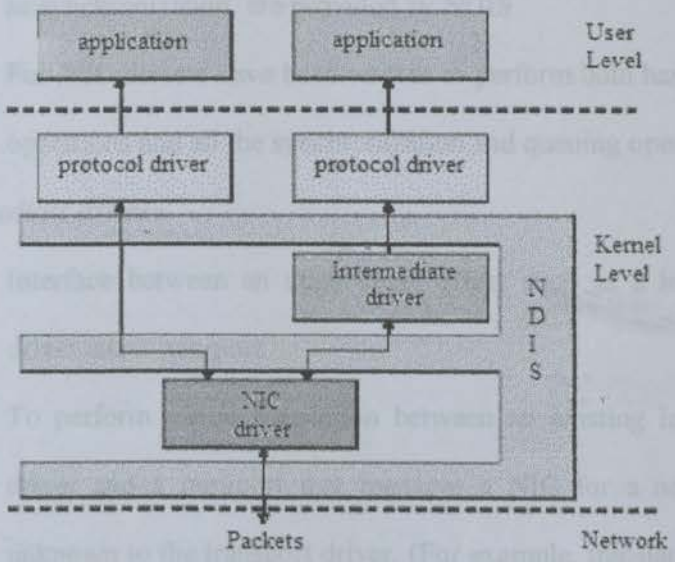


Figure 2.2.7.1: simple NDIS structure

NDIS support three types of network devices:

1. *Network interface card or NIC drivers*
   - NIC drivers directly manage network interface cards, referred to as NICs.
   - The NIC drivers interface directly to the hardware at their lower edge and at their upper edge present an interface to allow upper layers to send packets on the network, to handle interrupts, to reset the NIC, to halt the NIC and to query and set the operational characteristics of the driver.

- A NIC driver cannot communicate with user-mode applications, but only with NDIS intermediate drivers or protocol drivers. NIC drivers can be either miniports or legacy full NIC drivers.
- Miniport drivers implement only the hardware-specific operations necessary to manage a NIC, including sending and receiving data on the NIC. Operations that common to all lowest level NIC drivers, such as synchronization, are provided by NDIS.
- Full NIC drivers have been written to perform both hardware-specific operations and all the synchronization and queuing operations

2. *Intermediate drivers*
   - Interface between an upper-level driver such as a legacy transport driver and a miniport.
   - To perform media translation between an existing legacy transport driver and a miniport that manages a NIC for a new media type unknown to the transport driver. (For example, translate from LAN to ATM).
   - An intermediate driver cannot communicate with user-mode applications, but only with other NDIS drivers.
3. *Transport drivers or protocol drivers*
   - A protocol driver implements a network protocol stack such as IPX/SPX or TCP/IP, offering its services over one or more network interface cards.

2.2.8 WinPcap [9]

WinPcap is an architecture that adds to the operating systems of the Win32 family the ability to capture the data of a network using the network adapter of the machine. Moreover, it provides to the applications a high level API that makes simpler the use of its low-level capabilities. It is subdivided into three separate components: a *packet capture device driver*, a *low-level dynamic library* and a *high level static library*.

We shall look at these components in the structure of capture stack.

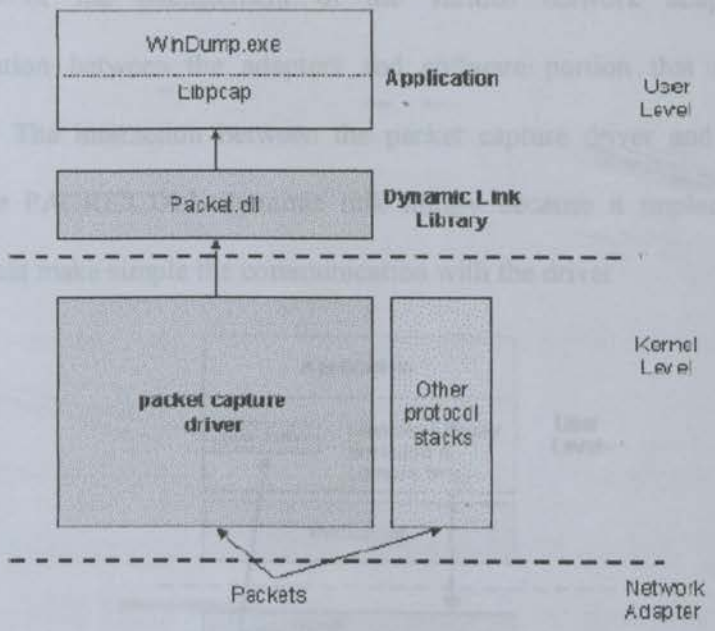


Figure 2.2.8.1: structure of capture stack

The user level capture application receives packets from the system, interprets, processes and output them to the user in a comprehensible and productive way. The pcap/ libpcap is a static library used by the packet capture part of the application. It provides a set of functions independents from the hardware and the operating system that an application can use to capture packets from a network. These functions include capture packets, set packet filters and communicate with the

network adapter. The PACKET.DLL is a dynamic link library that providing a system-independent capture interface. It also provides services to pcap static library.

The purpose of the kernel level is to take the link-layer packets from the network and to transfer them to the application level without modification. The Packet Capture Driver works at this level. It supplies the applications a set of the functions used to write and read data from the network at data-link layer. The packet capture driver interacts with the NICs through NDIS.

As I mention previously, The NDIS is a part of network code of Win32 that is responsible of the management of the various network adapters and the communication between the adapters and software portion that implement the protocols. The interaction between the packet capture driver and application is through the PACKET.DLL dynamic link library because it implements a set of functions that make simple the communication with the driver.

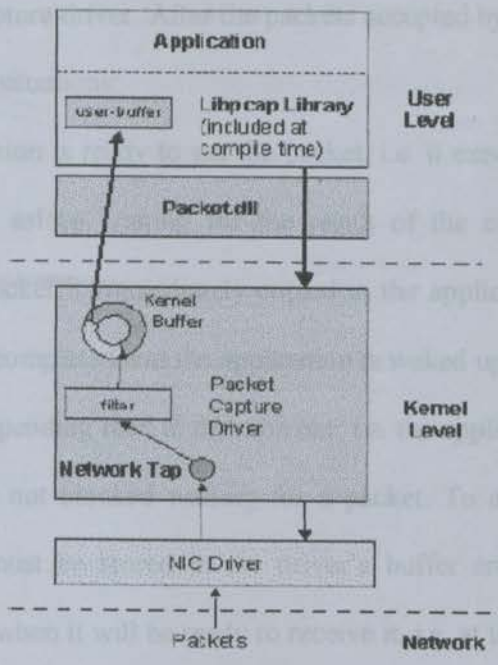


Figure 2.2.8.2: structure of driver

Figure 2.2.8.2 shows the processes of capturing packets by the winpcap, the arrow toward the top of the picture represent the flow of packets from the network to the capture application. The bigger arrow between the kernel buffer and the application indicates that more than one packet can transit between these two entities in a single read system call. The network tap is a callback function that is invoked by the network adapter's device driver when a new packet arrives. It collects the copies of packets from the NIC driver and delivers them to filter.

The filtering process is done at kernel level. If the packets satisfy the filter, it will be copied to application or put in the buffer if the application is not ready to receive it. If no filter defined, it will accept all packets. Filter is applied to NIC driver's memory, without copying to the packet capture driver. This allows the reject a packet before any copy and thus minimizing the load on the system.

The application obtain the packets from the network by perform the read call on the NDIS packet capture driver. After the packets accepted by the filter, the driver can be in two different situations:

- The application is ready to get the packet, i.e. it executed a read call and is currently asleep waiting for the result of the call. In this case the incoming packet is immediately copied in the application's memory, the read call is completed and the application is waked up.
- There is no pending read at the moment, i.e. the application is doing other stuff and is not blocked waiting for a packet. To avoid the loss of the packet, it must be stored in the driver's buffer and transferred to the application when it will be ready to receive it, i.e. at the next read call.

The driver uses a circular buffer to store the packets. A packet is stored in the buffer with a header that maintains information like the timestamp and the size of the

packet. Moreover, a padding is inserted between the packets in order to word-align them to increase the speed of the copies.

Incoming packets are discarded by the driver if the buffer is full when a new packet arrives. If the buffer is not empty when the application performs a read system call, the packets in the driver's buffer are copied to the application memory and the read call is immediately completed. More than one packet can be copied from the driver's circular buffer to the application with a single read call. This improves the performances because it minimizes the number of reads. To maintain packet boundaries, the driver encapsulates the captured data from each packet with a header that includes a timestamp and length.

The circular buffer method uses the memory more efficiently with big buffers because the free memory is always available. The aim circular buffer is to store the incoming packets and pass it to user-level buffer when the user level buffer is empty so the packets loss can be avoided since the application may not ready to read the packets that come in burst. The size of the circular buffer is bigger than the application's buffer, so the packet capture driver must detects the dimension of the application's buffer and fills it in the right way. When the dimension of the application's buffer is smaller than the number of bytes in the driver's buffer, the driver has to scan the headers of the packets in its buffer in order to determine the amount of bytes to copy. This process slows down a bit the capture process, therefore best performance are obtained when application and kernel buffers have the same size.

2.2.9 TCP/IP model [10]

Although the OSI reference model is universally recognized, the historical and technical open standard of the Internet is Transmission Control Protocol/Internet Protocol (TCP/IP). The TCP/IP reference model and the TCP/IP protocol stack make data communication possible between any two computers, anywhere in the world. Normally, networking protocols consist of different layers, with each layer responsible for a different aspect of the communication. The TCP/IP protocol suite is the combination of different protocols at various layers. TCP/IP is normally considered to be comprised of 4 layers, where each layer has its own specific responsibility

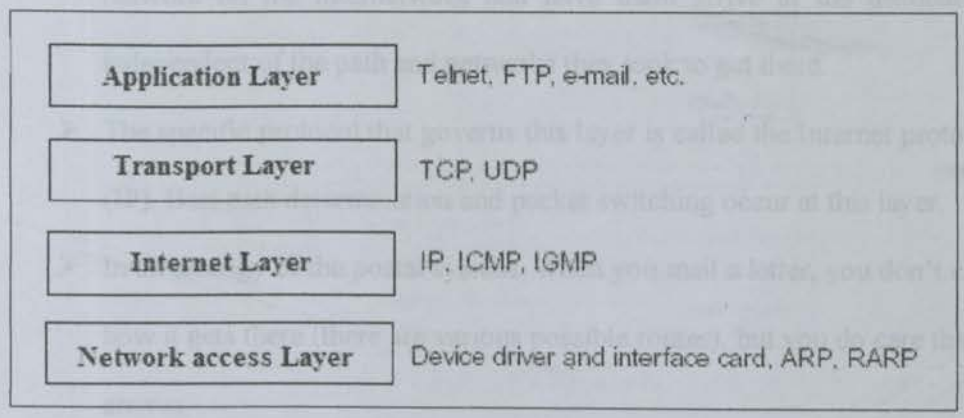


Figure 2.2.9.1: TCP/IP model

Application Layer

- Handles high level protocols, issues of representation, encoding, and dialog control.
- The TCP/IP combines all issues related to application into one layer, and assures this data is properly packaged for the next layer.

Transport Layer

- Deals with the quality of service issues of reliability, flow control, and error correction.

- The transmission control protocol (TCP), provides excellent and flexible ways to create reliable, well-flowing, low-error network communications.
- TCP is a connection-oriented protocol. It dialogues between source and destination while packaging application layer information into units called segments.
- Connection-oriented means that Layer 4 segments travel back and forth between two hosts to acknowledge the connection exists logically for some period (packet switching).

Internet Layer

- The purpose of the internet layer is to send source packets from any network on the internetwork and have them arrive at the destination independent of the path and networks they took to get there.
- The specific protocol that governs this layer is called the Internet protocol (IP). Best path determination and packet switching occur at this layer.
- In an analogy of the postal system. When you mail a letter, you don't care how it gets there (there are various possible routes), but you do care that it arrives.

Network Access Layer

- The name of this layer is very broad and somewhat confusing. It is also called the host-to-network layer.
- It is the layer that is concerned with all of the issues that an IP packet requires to actually make a physical link. It includes the LAN and WAN technology details, and all the details in the OSI physical and data link layers.

In the TCP/IP model, regardless of which application requests network services, and regardless of which transport protocol is used, there is only one network protocol - internet protocol, or IP. This is a deliberate design decision. *IP* serves as a universal protocol that allows any computer, anywhere, to communicate at any time.

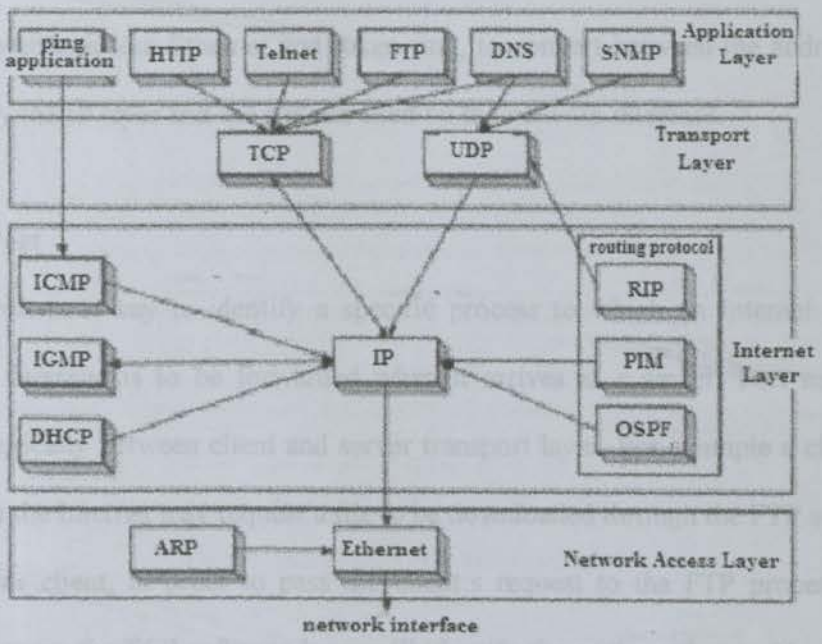


Figure 2.2.9.2: TCP/IP model

There are many different protocols in the TCP/IP protocol suite.

- TCP is one of the two predominant transport layer protocols. It uses IP as the network layer. TCP provide reliable transmission of data over a logical connection to HTTP
- IP is the main protocol at the internet layer. Every piece of data that gets transferred around an internet layer goes through the IP layer at both end systems and at every intermediate router. IP provide unreliable transmission of IP datagram across an IP network.

- ICMP is an attachment to IP. It is used by the IP layer to exchange error messages and other vital information with the IP layer in another host or router.
- Ethernet provide service to IP by giving transmission of a frame across an Ethernet segment for IP.
- ARP is a specialized protocol used only with certain types of network interfaces like Ethernet and token ring, to convert between the address used by the IP layer and the address used by the network interface.

2.2.10 Port

Port is a way to identify a specific process to which an Internet or other network message is to be forwarded when it arrives at a server. Port number is passed logically between client and server transport layer. For example a client to a server on the Internet may request a file to be downloaded through the FTP server.

For client, in order to pass the client's request to the FTP process in the remote server, the TCP software layer will identify the port number as 21 which are associated with an FTP request and append to client's request. At the server, the TCP layer will read the port number of 21 from the TCP header and forward the client's request to the FTP programs.

2.2.11 Encapsulation

When sending data using TCP, the data is sent down the protocol stack of the operating system, passing through each one of its layers, until it is finally sent as a stream of bits across the network. At each layer information is added to the data by prepending headers to the data that is received. The unit of data that TCP sends is called a TCP segment. The unit of data that IP sends to the network interface is

called an IP datagram. The stream of bits that flows across the Ethernet is called a frame (packet). TCP and ICMP send data to IP that forwards it to the network interface. The network interface sends and receives frames on behalf of IP, ARP, etc

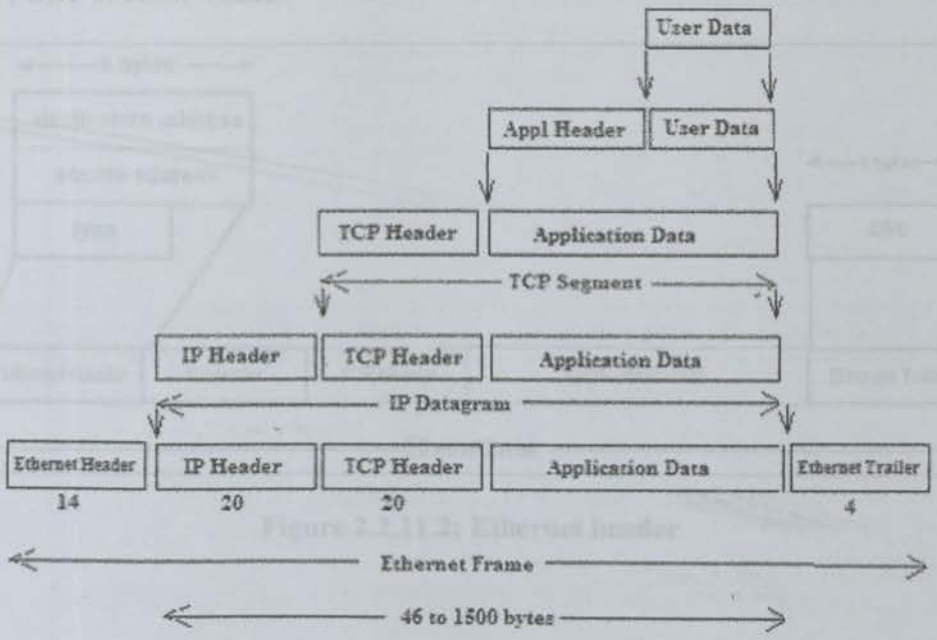


Figure 2.2.11.1: Data encapsulation

2.2.11.1 Ethernet frame

The Ethernet header contains the physical destination and source address which is each 6 bytes and the type field is 2 bytes. The destination MAC address is used by NIC's hardware filter to check whether it match its own MAC address and so to decide whether to receive the packet or not. The CRC in Ethernet trailer is 4 bytes.

IP datagram is divided to two main parts, IP header and the IP data which consists of TCP header and application data. As the figure shown, the IP header is in

5 layers with the length 32 bits (4 bytes) each that add up to total of 20 bytes or 160 bits.

Inside the IP data contain the 20 bits header of the segment which may be TCP, UDP or ICMP header.

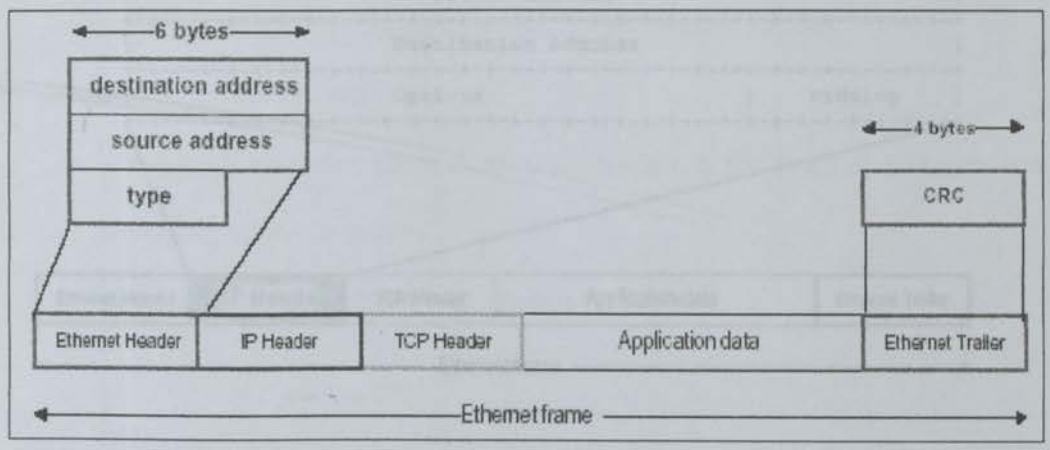


Figure 2.2.11.2: Ethernet header

2.2.12 Internet Protocol (IP) [11]

The internet protocol's basic function is deal with addressing and fragmentation. The internet module use the address carried in the IP header to transmit datagram towards their destination. The other fields in the IP header are used to fragment and reassemble IP datagram to small packets.

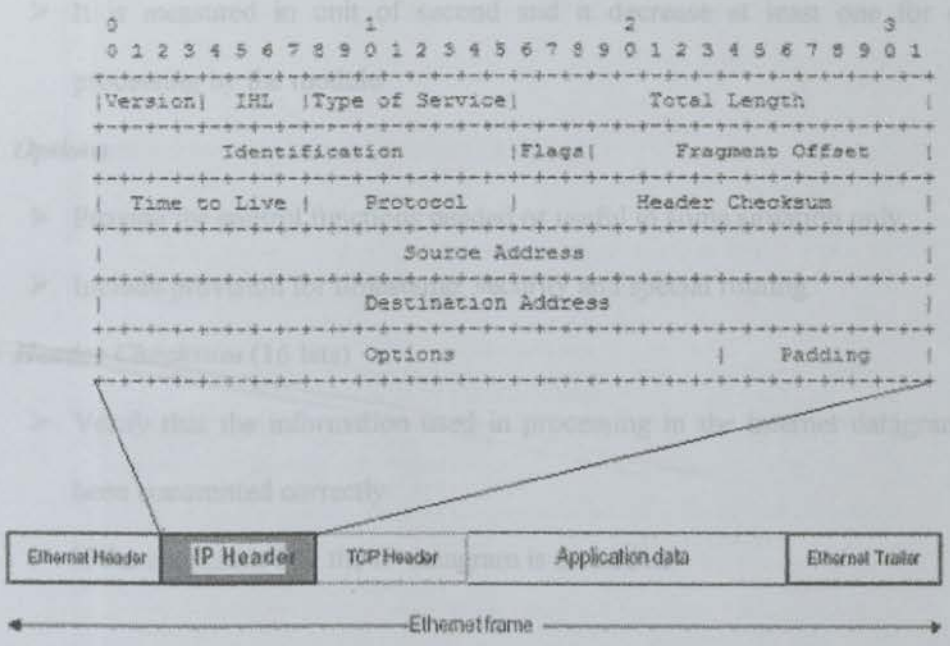


Figure 2.2.12: IP header

The IP protocol uses 4 key mechanisms in providing its service:

1. ***Type of service*** (8 bits)

   - The Type of Service provides an indication of the abstract parameters of the quality of service desired. Several network offer service precedence, which in some way it treat high precedence traffic as more important than other traffic.
   - Thus these parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network.
   - The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput.

2. ***Time to Live*** (8 bits)

   - Indicates the maximum time the datagram is allowed to stay in the internet system so that undelivered datagram can be discarded.

- It is measured in unit of second and it decrease at least one for every processing by the module.

3. ***Options***
   - Provide for control functions needed or useful in some situation only.
   - Include provision for timestamp, security and special routing.
4. ***Header Checksum*** (16 bits)
   - Verify that the information used in processing in the internet datagram has been transmitted correctly.
   - If the checksum fail, the IP datagram is discarded.

Other fields:

1. ***Version*** (4 bits) – indicate format of the IP header.
2. ***IHL*** (4 bits) – length of the internet header, usually is 5 and minimum value for a correct header is 5.
3. ***Total length*** (16 bits) – total length of IP datagram (including IP header and IP data). Maximum length is up to 2^{16}-1=65535.
4. ***Identification*** (16 bits) – an identifying value assigned by the sender to aid in assembling the fragments of datagram.
5. ***Flags*** (3 bits) – contain various control flag associate with fragments.
6. ***Fragment offset*** (13 bits) – indicates where in the datagram this fragment belongs.
7. ***Protocol*** (8 bits) – indicates the next level protocol used in the data portion of the internet diagram. The protocol number 1 indicates ICMP, 2 for IGMP, 6 for TCP, and 17 for UDP.
8. ***Source address*** (32bits) – source IP address of packet.

9. ***Destination address*** (32bits) – destination IP address of packet.

2.2.13 Transmission Control Protocol (TCP) [12]

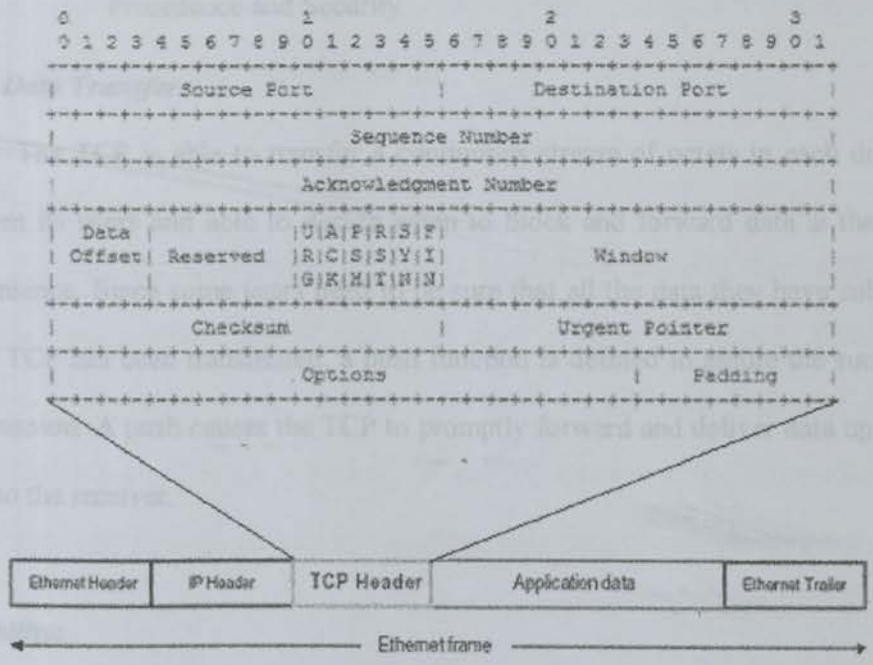


Figure 2.2.13: TCP header

The TCP is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.

The interface between an application process and the TCP is much like the calls an operating system provides to an application process for manipulating files. There are calls to open and close connections and to send and receive data on established connections. The primary purpose of the TCP is to provide reliable, securable logical circuit or connection service between pairs of processes and this requires facilities in the following areas:

Basic Data Transfer

Reliability
Flow Control
Multiplexing
Connections
Precedence and Security

Basic Data Transfer:

The TCP is able to transfer a continuous stream of octets in each direction between its users and able to decide when to block and forward data at their own convenience. Since some users need to be sure that all the data they have submitted to the TCP has been transmitted, a push function is defined to assure the successful transmission. A push causes the TCP to promptly forward and deliver data up to that point to the receiver.

Reliability:

The TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by the internet communication system. This is achieved by assigning a sequence number to each octet transmitted, and requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver, the sequence numbers are used to correctly order segments that may be received out of order and to eliminate duplicates. Damage is handled by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments. As long as the TCPs continue to function properly and the internet system does not become completely partitioned, no transmission errors will affect the correct delivery of data.

Flow Control:

TCP provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission.

Multiplexing:

To allow for many processes within a single Host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. The concatenation with the network and host addresses from the internet communication layer forms a socket. A pair of sockets uniquely identifies each connection. That is, a socket may be simultaneously used in multiple connections.

Connections:

The reliability and flow control mechanisms described above require that TCP initialize and maintain certain status information for each data stream. The combination of this information, including sockets, sequence numbers, and window sizes, is called a connection. Each connection is uniquely specified by a pair of sockets identifying its two sides. When two processes wish to communicate, their TCP's must first establish a connection (initialize the status information on each side). When their communication is complete, the connection is terminated or closed to free the resources for other uses. Since connections must be established between unreliable hosts and over the unreliable internet communication system, a handshake mechanism with clock-based sequence numbers is used to avoid erroneous initialization of connections.

Precedence and Security:

The users of TCP may indicate the security and precedence of their communication. Provision is made for default values to be used when these features are not needed.

TCP segment are sent as IP datagram. Like IP header, a TCP header also supplying information specific to the TCP itself. The fields of the header are:

1. ***Source port*** (16 bits)

- indicate the source port number

2.***destination port*** (16 bits)

- indicate the destination port number

3.***sequence number*** (32 bits)

- Every octet data sent over a TCP connection has a sequence number so that every octet is sequenced. Each of them thus must be acknowledged through a cumulative acknowledgement mechanism.
- this field indicate the first data octet in this segment (when the SYN is not present)
- If the SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

4.***acknowledged number*** (32 bits)

- If the ACK control bit is set, this field will contains the value of the next sequence number of the sender of the segment is expecting to receive.
- Once a connection is established this is always sent.
- An acknowledgment of sequence number X indicates that all octets up to but not including X have been received.

5.***Data offset*** (4 bits)

- indicate where the data begins

6. ***Control bits*** (6 bits)

- URG : urgent pointer field significant
- ACK : acknowledgement field significant
- PSH : push function
- RST : reset the connection
- SYN : synchronize sequence number
- FIN : no more data from sender

7. ***windows*** (16 bits)

- specifies the number of octets, starting with the acknowledgment number, that the receiving TCP is currently prepared to receive

8. ***checksum*** (16 bits)

- The checksum field is the 16 bits one's complement of the one's complement sum of the all 16 bit words in the header and text.

9. ***Urgent pointer*** (16 bits)

- This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment.
- The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

10. ***Options*** (16 bits)

- Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length.

2.2.13.1 Three Way Handshakes

For a connection to be established or initialized, the two TCPs must synchronize on each other's initial sequence numbers (SEQ). This is done in an exchange of connection establishing segments carrying a control bit called "SYN" (synchronize) and the initial sequence numbers (ISN). As shorthand, segments carrying the SYN bit are also called "SYNs". Hence, the solution requires a suitable mechanism for picking an initial sequence number and a slightly involved handshake to exchange the ISN's.

The synchronization requires each side to send its own initial sequence number and to receive a confirmation of it in acknowledgment from the other side. Each side must also receive the other side's initial sequence number and send a confirming acknowledgment.

1) A --> B SYN my sequence number is X

2) A <-- B ACK your sequence number is X

3) A <-- B SYN my sequence number is Y

4) A --> B ACK your sequence number is Y

A send a SYN to B to tell its sequence number (SEQ) is X. B accept and acknowledge the SYN for A is X and at the same time send its SEQ, Y to A. A then reply the acknowledgement of B's SEQ. Because steps 2 and 3 can be combined in a single message this is called the three way handshake.

A three way handshake is necessary because sequence numbers are not tied to a global clock in the network, and TCPs may have different mechanisms for picking the ISN's. The receiver of the first SYN has no way of knowing whether the segment was an old delayed one or not, unless it remembers the last sequence number used on

the connection (which is not always possible), and thus it must ask the sender to verify this SYN.

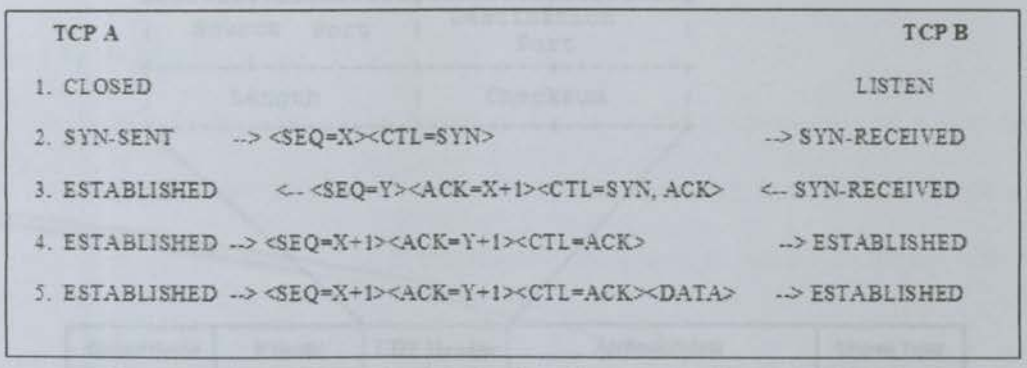


Figure 2.2.13.1 Way Handshake for Connection Synchronization

In line 2 TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting X. In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence X+1, acknowledging the SYN which occupied sequence X. At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN; and in line 5, TCP A sends some data. Note that the *sequence number* of the segment in line 5 is the same as in line 4 because the ACK DOES NOT occupies sequence number space.

2.2.14 User Datagram Protocol (UDP) [13]

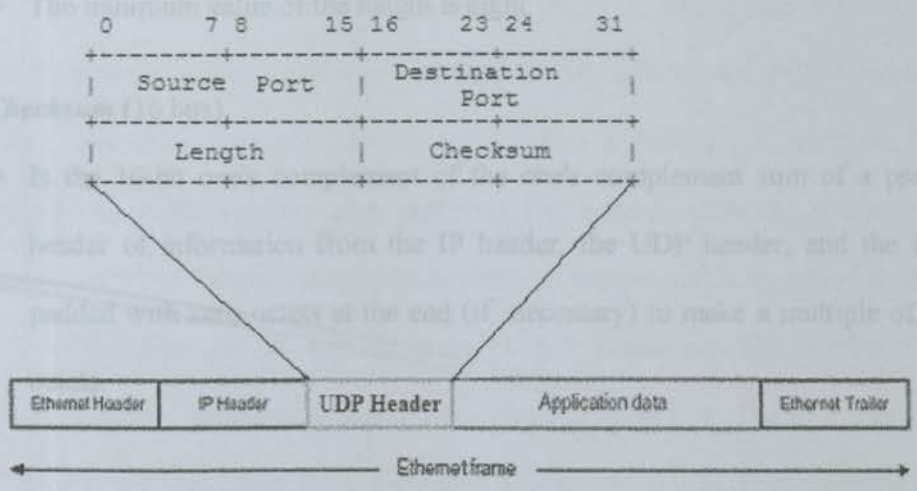


Figure 2.2.14: UDP header

UDP is another protocol at transport layer which is transaction oriented, and delivery and duplicate protection are not guaranteed. While TCP is used for highly reliable host-to-host packet-switched computer communication, UDP provides a procedure for application programs to send messages to other host with a minimum used of protocol mechanism.

1. ***source port*** (16 bits)
   - Is an optional field, when meaningful, it indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information.
   - If not used, a value of zero is inserted.
2. ***destination port*** (16 bits)
   - Is has a meaning within the context of a particular internet destination address.
3. ***Length*** (16 bits)

- The length in octets of this user datagram including this header and the data.
- The minimum value of the length is eight.

4. ***Checksum*** (16 bits)
   - Is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

2.2.15 Address Resolution Protocol (ARP) [14]

ARP is a protocol for mapping an IP address to a physical machine address that is recognized in the local network. For example when an incoming packet destined for a host machine on a particular local area network arrives at a gateway, the gateway asks the ARP program to find a physical host or MAC address that matches the IP address. The ARP program looks in the ARP cache and, if it finds the address, provides it so that the packet can be converted to the right packet length and format and sent to the machine. If not, ARP will broadcasts a request packet to all the machines on the LAN to see if one machine knows that it has that IP address associated with it. A machine that recognizes the IP address as its own returns a reply so indicating.

ARP header

ARP header has the following fields:

Fields	Length	description
HRD	16 bit	Hardware type.
PRO	16 bit	Protocol type. 0x800 for IP
HLN	8 bit	Hardware address length. usually is 6
PLN	8 bit	Protocol address length. usually is 4
OP	16 bit	Opcode. 1 for request and 2 for reply.

Table 2.2.15: ARP header

2.2.16 Internet Control Message Protocol (ICMP) [15]

ICMP is used by Hosts and gateways as a messaging, control, and diagnostic protocol to alert a host of problems or test connectivity. ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route.

ICMP messages are sent using the basic IP header and IP identifies ICMP messages contained within an IP datagram with protocol type 1. The first four bytes of ICMP header (1-byte type field, 1-byte code field, and 2-byte checksum) have the same format for all message types. All other fields and information contained within the ICMP header vary depending on the message type being sent. The type field identifies the type of the message sent by the host or gateway.

Type	Description ICMP Message Types
0	Echo Reply (Ping Reply, used with Type 8, Ping Request)
3	Destination Unreachable
4	Source Quench
5	Redirect
8	Echo Request (Ping Request, used with Type 0, Ping Reply)
9	Router Advertisement (Used with Type 9)
10	Router Solicitation (Used with Type 10)
11	Time Exceeded
12	Parameter Problem
13	Timestamp Request (Used with Type 14)
14	Timestamp Reply (Used with Type 13)
15	Information Request (obsolete) (Used with Type 16)
16	Information Reply (obsolete) (Used with Type 15)
17	Address Mask Request (Used with Type 17)
18	Address Mask Reply (Used with Type 18)

Table 2.2.15: ICMP message type

ICMP messages have two types of codes (query and error). The error types are:

- Type 3 = Destination Unreachable
- Type 4 = Source Quench
- Type 5 = Redirect
- Type 11 = Time Exceeded
- Type 12 = Parameter Problems

Each error types have their own code number to specifically identify the information about the error condition. Queries type message always contain no additional information because they merely ask for information and will show a value of 0 in the code field.

The checksum verifies the validity of the ICMP header. The sending host performs the initial checksum calculation and places the results in this field. The receiving host performs the same calculations to assure that it does not receive data damaged in transit. If the checksum values do not match, it trashes the datagram.

2.2.17 Packet analyzing

Recall that in encapsulation, the data is sent down the through each one of its TCP/IP layers from application until it is finally sent as a stream of bits across the network. At each layer information is added to the data. Therefore we can trace the fields of the packets beginning from the Ethernet header (data-link layer) until application layer to analyze the packets into human readable format. The data in the lower layer fields of the packet will identify the protocol and format for the next upper layer of data. For example, if the value of protocol field in IP header is 6, we know that the protocol used in transport layer is TCP.

It is well-known that in fact computers only handle **0**s (zeros) and **1** (ones). By means of sequences of **0** and **1** the computer can express in binary form diverse ranks of numbers. However there is no such way to represent letters with **0**s and **1**s. So, computers use the ASCII code, that is a table or list that contains all the letters in the alphabet plus other additional characters. In this code each character is always represented by the same order of Hexadecimal number. For example, in ASCII code

the capital letter A is always represented by the order number 65, which is easily represented in binary as **1000001**.

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

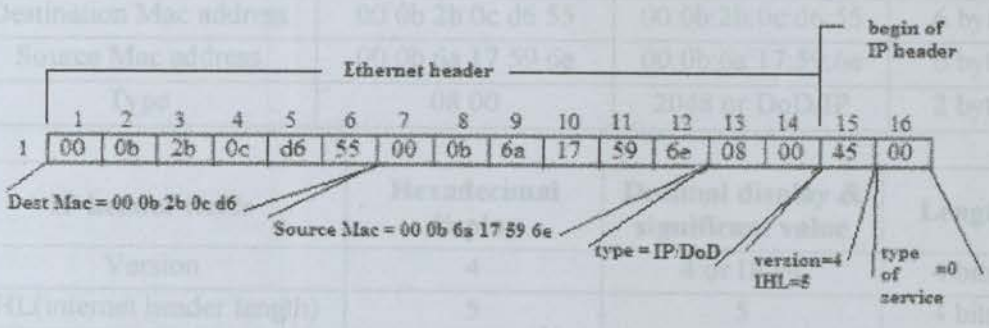
Table 2.2.17.1: ASCII code table [16]

Example 1: TCP packet with frame length = 62 byte

Hex display of Frame

```
0x0000 00 0b 2b 0c d6 55 00 0b 6a 17 59 6e 08 00 45 00   ..+..U..j.Yn..E.
0x0010 00 30 09 b1 40 00 80 06 04 1d 7b 7b 7b 02 7b 7b   .0..@.....{{{.{{
0x0020 7b 01 04 20 00 50 42 d3 03 77 00 00 00 01 70 02   {.. .PB..w....p.
0x0030  ff  ff  4b 6b 00 00 02 04 05 b4 01 01 04 02          ..Kk..........
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	00	0b	2b	0c	d6	55	00	0b	6a	17	59	6e	08	00	45	00
2	00	30	09	b1	40	00	80	06	04	1d	7b	7b	7b	02	7b	7b
3	7b	01	04	20	00	50	42	d3	03	77	00	00	00	01	70	02
4	ff	ff	4b	6b	00	00	02	04	05	b4	01	01	04	02		



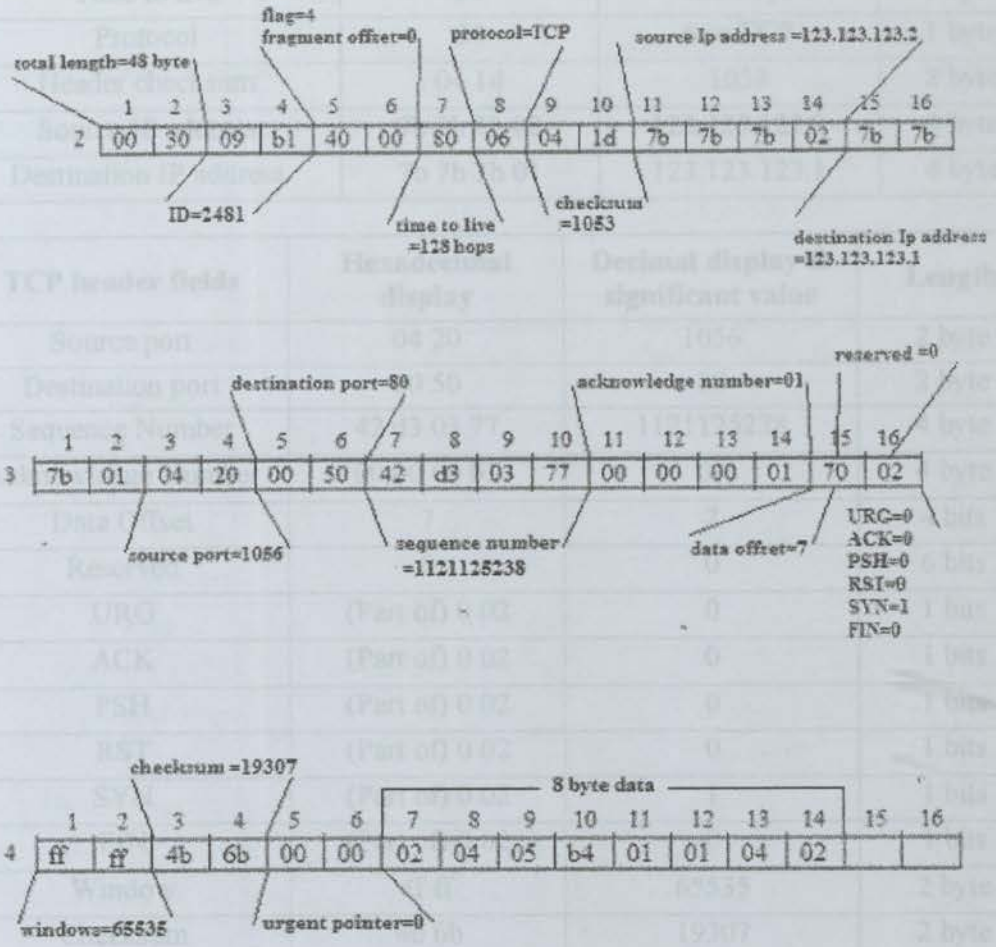


Figure 2.2.17: Analyzing of frame

Ethernet header Fields	Hexadecimal display	Decimal display & significant value	Length
Destination Mac address	00 0b 2b 0c d6 55	00:0b:2b:0c:d6:55	6 byte
Source Mac address	00 0b 6a 17 59 6e	00:0b:6a:17:59:6e	6 byte
Type	08 00	2048 or DoD/IP	2 byte

IP header fields	Hexadecimal display	Decimal display & significant value	Length
Version	4	4 or IP v4	4 bits
IHL(internet header length)	5	5	4 bits
Type of service	00	0	1 byte
Total length	00 30	48 byte	2 byte
Identification	09 b1	2481	2 byte
Flag	(Part of) 40 00	2	3 bits
Segment Offset	(Part of) 40 00	0	13 bits

Time to live	80	128 hops	1 byte
Protocol	06	6 or TCP	1 byte
Header checksum	04 1d	1053	2 byte
Source IP address	7b 7b 7b 02	123.123.123.2	4 byte
Destination IP address	7b 7b 7b 01	123.123.123.1	4 byte

TCP header fields	Hexadecimal display	Decimal display & significant value	Length
Source port	04 20	1056	2 byte
Destination port	00 50	80	2 byte
Sequence Number	42 d3 03 77	1121125238	4 byte
Acknowledge Number	00 00 00 01	01	4 byte
Data Offset	7	7	4 bits
Reserved	0	0	6 bits
URG	(Part of) 0 02	0	1 bits
ACK	(Part of) 0 02	0	1 bits
PSH	(Part of) 0 02	0	1 bits
RST	(Part of) 0 02	0	1 bits
SYN	(Part of) 0 02	1	1 bits
FIN	(Part of) 0 02	0	1 bits
Window	ff ff	65535	2 byte
Checksum	4b 6b	19307	2 byte
Urgent pointer	00 00	0	2 byte

Table 2.2.17.2: Protocol fields of TCP packet

Example 2: UDP packet with frame length = 62 byte

Hex display of Frame

```
0000  01 00 5E 00 00 02 00 04 C0     ....^.....À¾
0009  F8 02 44 08 00 45 C0 00 30     ø.D..E À...0
0012  00 00 00 00 02 11 9F 85 CA     .....:...Ÿ...Ê
001B  B9 6D BC E0 00 00 02 07 C1     ¹m¼à.....Á
0024  07 C1 00 1C 47 05 00 00 08     ... Á..G.....
002D  03 0A 6E 00 00 63 69 73 63     ......n...cisc
0036  6F 00 00 00 CA B9 6D BE        o.....Ê¹m¾
```

Ethernet header Fields	Hexadecimal display	Decimal display & significant value	Length
Destination Mac address	01 00 5E 00 00 02	01:00:5E:00:00:02	6 byte
Source Mac address	00 04 C0 F8 02 44	00:04:C0:F8:02:44	6 byte
Type	08 00	2048 or DoD/IP	2 byte

IP header fields	Hexadecimal display	Decimal display & significant value	Length
Version	4	4 or IP v4	4 bits
IHL(internet header length)	5	5	4 bits
Type of service	C0	192	1 byte
Total length	00 30	48 byte	2 byte
Identification	00 00	0	2 byte
Flag	(Part of) 00 00	0	3 bits
Segment Offset	(Part of) 00 00	0	13 bits
Time to live	02	2 hops	1 byte
Protocol	11	17 or UDP	1 byte
Header checksum	9F 85	40837	2 byte
Source IP address	CA B9 6D BC	202.185.109.188	4 byte
Destination IP address	E0 00 00 02	224.0.0.2	4 byte

UDP header fields	Hexadecimal display	Decimal display & significant value	Length
Source port	07 C1	1985	2 byte
Destination port	07 C1	1985	2 byte
Length	00 1C	28 byte	2 byte
Checksum	47 05	18181	2 byte

Table 2.2.17.3: Protocol fields of UDP packet

Chapter 3

Software Architecture

Chapter 3 – Software Architecture

3.1 Language

3.1.1 C

C programming language was developed at Bell Labs during the early 1970's. C is rather like Pascal or FORTRAN. C is a structure programming language where values are stored in variables. Programs are structured by defining and calling functions. Program flow is controlled using loops, if statements and function calls. Input and output can be directed to the terminal or to files. Related data can be stored together in arrays or structures.

3.1.2 C++

C++ programming language is an object-oriented programming. C++ is replacing the more traditional structured programming techniques in C. Since C++ is superset of C it gains many of the attractive features of the C language, such as efficiency, closeness to the machine, and a variety of built-in types. A number of new features were added to C++ to make the language even more robust, many of which are not used by novice programmers.

3.1.3 Visual Basic

Visual Basic is both programming language and environment developed by Microsoft and since its launch in 1990 it has become the standard for programming languages. Based on the BASIC language, Visual Basic was the first products to provide

a graphical programming environment and a paint metaphor for developing user interfaces. The Visual Basic programmer can add a substantial amount of code simply by dragging and dropping *controls*, such as buttons and dialog boxes, and then defining their appearance and behavior instead of worrying of about the syntax details.

Now there are visual environments for many programming languages, including C, C++, Pascal, and Java. Visual Basic is sometimes called a Rapid Application Development (RAD) system because it enables programmers to quickly build prototype applications.

3.1.4 Java

Java is an object-oriented programming language with a built-in application programming interface (API) that can handle graphics and user interfaces. It can be used to create applications or applets. Since its rich set of API's, similar to Macintosh and Windows, and its platform independence, Java can also be thought of as a platform in itself. Java also has standard libraries for doing mathematics.

Java's syntax most is the same as C and C++ but the major difference is that Java does not have pointers like in C or C++ and you must write object oriented code in Java. Procedural pieces of code can only be embedded in objects.

In Java we distinguish between applications, which are programs that perform the same functions as those written in other programming languages. Java has advanced features that it can be embedded in a Web page and accessed over the Internet with using applets.

3.1.5 C#

C# is a new language for Windows applications, intended as an alternative to the main previous languages, C++ and VB. C# is Developed at Microsoft by a team led by Anders Hejlsberg and Scott Wiltamuth. It incorporated into .NET platform which can use as Web based applications. C# also widely used in network programming by quickly prototype and deploy network applications using C# classes. The C# language, a close cousin to Java, is a new object-oriented programming language (OOPL) designed to work within the .NET framework. It improves upon many of the vague or ill-defined areas of C++ that frequently lead programmers into trouble. C# is a strongly-typed, object-oriented language designed to give the optimum blend of simplicity, expressiveness, and performance.

3.1.6 Comparison of C/C++, C# and JAVA Language

	Java	C#	C++
Object-Orientation	Hybrid	**Hybrid**	Hybrid / Multi-Paradigm
Static / Dynamic Typing	Static	**Static**	Static
Generic Classes	No	**No**	Yes
Inheritance	Single class, multiple interfaces	**Single class, multiple interfaces**	Multiple
Feature Renaming	No	**No**	No
Method Overloading	Yes	**Yes**	Yes

	Java	C#	C++
Operator Overloading	No	**Yes**	Yes
Higher Order Functions	No	**No**	No
Lexical Closures	No	**No**	No
Garbage Collection	Mark and Sweep or Generational	**Mark and Sweep or Generational**	None
Class Variables / Methods	Yes	**Yes**	Yes
Reflection	Yes	**Yes**	No
Access Control	public, protected, "package", private	**public, protected, private, internal, protected internal**	public, protected, private, "friends"
Design by Contract	No	**No**	No
Multithreading	Yes	**Yes**	Libraries
Regular Expressions	Standard Library	**Standard Library**	No
Pointer Arithmetic	No	**Yes**	Yes
Language Integration	C, some C++	**All .NET Languages**	C, Assembler
Built-In Security	Yes	**Yes**	No
Object-Oriented Features			
Encapsulation / Information Hiding	Yes	**Yes**	Yes

	Java	C#	C++
Inheritance	Yes	**Yes**	Yes
Polymorphism / Dynamic Binding	Yes	**Yes**	Yes
All pre-defined types are Objects	No	**No**	No
All operations are messages to Objects	No	**No**	No
All user-defined types are Objects	Yes	**Yes**	No

Table 3.1.6: Comparison among the three Languages

3.2 Authoring Tools

3.2.2 Microsoft Visual Studio 6.0

Microsoft Visual Studio 6.0 is the complete enterprise development tool suite used to rapidly build business solutions. Visual Studio allows you to build scalable applications for Windows and the Web that easily integrate with existing systems. The Visual Studio 6.0 development suite sets the standard for developer productivity and comprehensive design support with integrated features across all the popular languages. Visual Studio 6.0 Enterprise Edition includes the complete set of development tools for building reusable applications in Microsoft Visual Basic 6.0, Visual C++ 6.0, Visual J++

6.0, or Visual FoxPro 6.0. Visual InterDev 6.0 provides easy integration with the Internet and a full Web page authoring environment. In addition, Visual Studio Enterprise Edition adds extensive support for large systems and distributed applications.

3.2.2 Microsoft Visual Studio .NET 2003

Microsoft Visual Studio .NET 2003 is a software development tool designed for programmers to build a broad range of applications for Microsoft Windows, the Web, and mobile devices. Microsoft Visual Studio .NET 2003 enhances, further refines, and is highly compatible with its predecessor. With Visual Studio .NET 2003, the integrated development environment (IDE) provides a consistent interface for all languages, including Microsoft Visual Basic .NET, Microsoft Visual C++ .NET, Microsoft Visual C# .NET, and Microsoft Visual J# .NET. Using the language best suited to your skill set, you can take advantage of shared visual designers to build rich Windows-based applications and dynamic Web applications that render in any browser.

Microsoft Visual Studio .NET 2003 enables developers to build Internet applications. It also provides developers with the tools for integrating solutions across operating systems and languages. With Visual Studio .NET, developers can easily convert existing business logic into reusable XML Web Services, encapsulating processes and making them available to applications on any platform.

Pros: - offers multiple language support.

- supports the Microsoft .NET Framework, which provides the common language runtime and unified programming classes

- include MSDN Library, which contains all the documentation for these development tools
- intuitive tools for working with XML and XSD files
- cross-language, cross-process, and remote debugging
- statement completion and syntax notification for HTML and XML tags

2.5 Operating System

An operating system is a set of programs that controls the hardware of a computer how to works. An operating system deals with the loading of software and management of memory. The operating system enables applications to save files to disk or print them via the printer. The operating system also determines what applications software will run on it. For security, OS ensures that unauthorized users do not access the system. The most widely used operating systems are Windows, LINUX.

2.5.2 Linux

Linux is a free open source operating system originally developed by Linus Torvalds in the early 1990s. It was released as open source and remains a worldwide developed operating system mainly distributed via the Internet.

Linux become a popular operating system for Internet/ intranet serving purposes. With a host of performance enhancements that will benefit Web sites and Internet sites of all sizes. Linux has made progress, primarily in functionality important to Internet infrastructure and Web server capabilities, including a greater selection of drivers, easier

installation, and GUI-based front ends for Web administration and window management.

Pros: - better network management tools

- new support for SMP and RAID arrays
- improved compiler performance

Cons: - reliance on Red Hat tools to the detriment of system administrators

- Administration tools are less sophisticated than those found in more mature Unix version

2.5.4 Windows XP

Windows XP is Microsoft latest version of popular windows operating system. Windows XP has two version, home and professional version. Both of two versions have a sleek and clean new interface that is more customizable than before and it looks great, with rounded window corners, larger and more detailed icons, and a clean-look desktop that on first installation shows only the taskbar and recycle bin.

Both of two versions provide Fast User Switching is a great feature for computers used by more than one person. It lets another user log on without killing the other user's session. When you switch back, running applications and open documents are there as you left them. This is impressive, but what really counts is that *XP* properly understands how to deal with multiple users. Each user has their own special folders, like My Documents, which cannot be seen by other users.

XP professional version has best performance in LAN, the Network Setup wizard simplifies setting up a network, and there is built-in support for 802.11. This is

peer-to-peer networking, and you can also connect to a domain resource such as a shared directory or printer by entering a user name and password. Business users' note: unlike Windows 98 or ME, cannot join a Windows server domain, so the networking is peer-to-peer only.

Pros:
- provides better integration of Windows 9x and Window NT that did Windows 2000
- uses slightly more total memory for the OS to add features than does Windows 2000
- offers significant GUI enhancements
- built-in support for compresses files
- advanced file sorting options
- more stable and improve troubleshooting tools

Chapter 4

Methodology and System Analysis

Chapter 4 – Methodology and System Analysis

4.1 Methodology

It is important to have a good procedure of design process before start doing any software development project. The efficient development of project depends on thoroughly systematic planning of progress of the project. The system development methodology is a method to create a system with a series of steps or operations or can be defined as system life cycle model. Every system development process model includes system requirements (user, needs, resource) as input and a finished product as output.

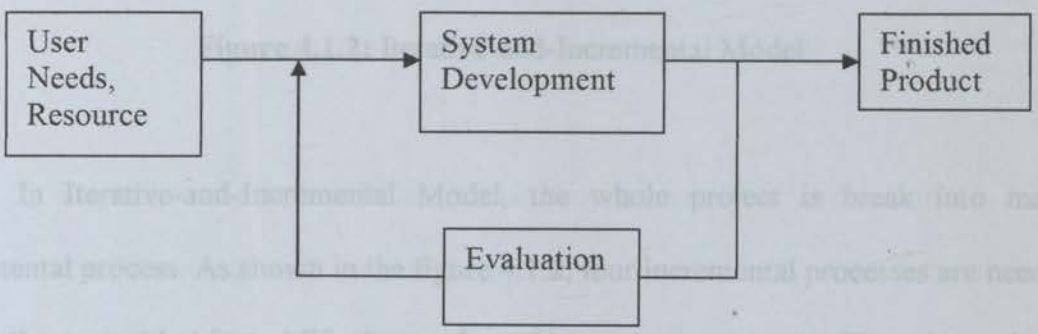


Figure 4.1.1 system Development Process Model

There are many life cycle models in real world such as waterfall model, spiral model, Rapid prototyping, iterative-and-incremental, etc. Among these model, Iterative-and incremental model is chosen.

4.1.1 Iterative and Incremental Model [17]

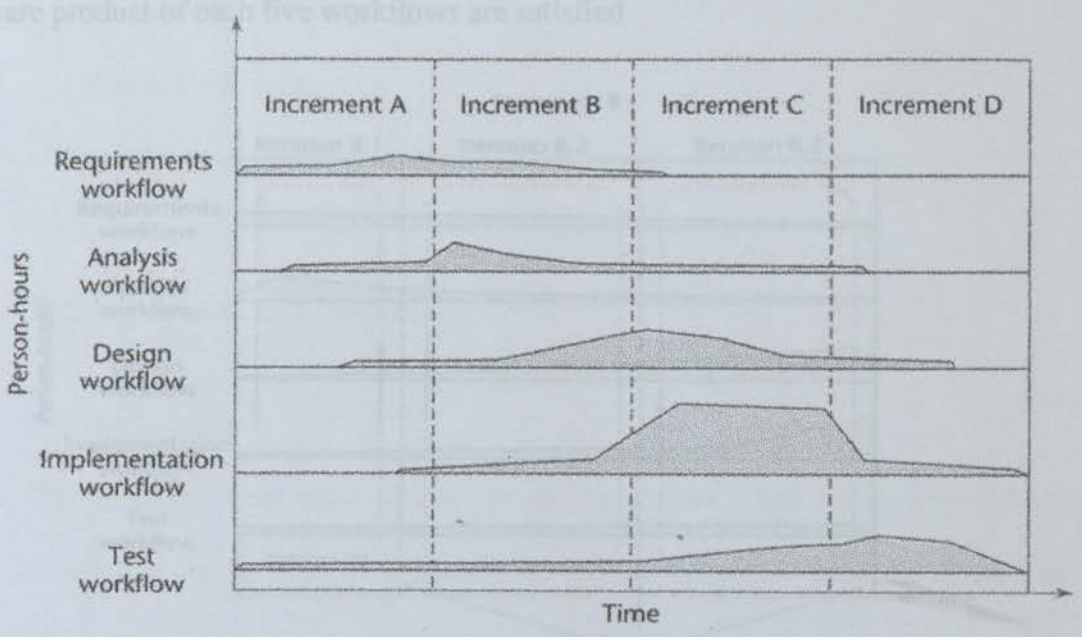


Figure 4.1.2: Iterative-and-Incremental Model

In Iterative-and-Incremental Model, the whole project is break into many incremental process. As shown in the figure 4.1.2, four incremental processes are needed to end the project but in real life the number of increments may vary. There is no single requirement phase, design phase or implementation phase as but in each five core workflow are performed in the entire life cycle, the different only at most times one workflow will predominates. At the beginning of the life cycle, the requirements workflow predominates and at the end of the life cycle the implementation workflow and test workflows predominate in the increment process.

Each increment has a number of iterations that performed together with five workflows (requirements, analysis, design, implementation and testing). There is no planning phase but the planning and documentation are carried out throughout the entire life cycle. Figure 4.1.3 depicts the enlargement of increment B in from the figure 4.1.2,

which has three iterations. Note that the number of iteration depends on when the software product of each five workflows are satisfied.

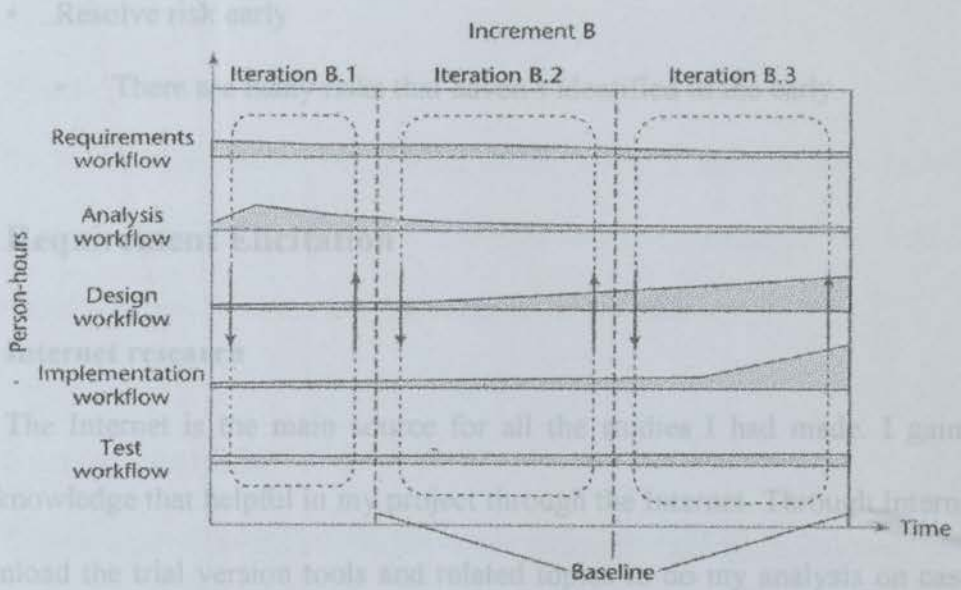


Figure 4.1.2: Iterative-and-Incremental Model

In Iterative-and Incremental model we can illustrate the whole project is break into a set of sub projects, with each of them is an increment. Each of the iteration has a loop back within the five workflows and so it can be viewed as a small and complete waterfall life cycle model. Thus Iterative-and Incremental model is actually consisting of many mini projects that implement the waterfall life cycle model.

Reasons for choosing Iterative-and-Incremental life cycle model:

- Multiple opportunities to check the correctness of the software product
    - Each of iteration offers a further opportunity to find faults and correct them.
    - Robustness of the architecture can be determined early

- The issue of integration and handlings of extension and changes is important in this project.

- Resolve risk early
  - There are many risks that haven't identified in the early.

4.2 Requirement Elicitation

4.2.1 Internet research

The Internet is the main source for all the studies I had made. I gained very useful knowledge that helpful in my project through the Internet. Through Internet I able to download the trial version tools and related topics to do my analysis on case study. Visiting the large corporate or organization's website like Microsoft.com may provide us the information on the latest software and technologies. Besides, Internet enables us to find information world widely, faster and easily and mostly not available on books.

4.2.2 Library and books

Library provides books that may use as a reference to my project. Books that related to software engineering and programming are very useful to refer. The documents in the FSKTM's documentation room had provided me the guideline and format on doing the documentation in my projects.

4.2.3 Interview with supervisor

Interview with supervisor had provided me the useful guideline and information on doing the research on my project. He defines the scopes and idea of the project which

I never think of. The comments and advice from him enables me to further renovate the project.

4.2.4 Practical testing

To see exactly how the real packet sniffer works, I had tested the tools in a Lab where I have permission to install the software and runs in the real network environment. The results of the test had reported in my case study.

4.3 System analysis

System analysis is the evaluation of the alternative solutions and specifications of detailed computer-based solutions. Before the software is designed the requirements and its specification is defined. The type of software used would be defined here.

4.4 Functional Requirements

Functional requirement is a statement of the service or functions that a system should provide how the system reacts to particular inputs, and how the system should behave in particular situations. It depends on the type of software, expected users and the type of system where the software is used. After the analysis done on the case study I was able to figure out the modules of the project.

4.4.1 Packet Capturing Module

This module establishes the packet capturing in the promiscuous mode. The configuration on the memory allocated, interface used, capturing option like stop capture

when reach a number of packet or bytes will be issued in this module. It should have the "start", "stop" and "continue" button to handle the capturing process. Besides the "save" function should exist to enable the saving of packets captured into a notepad file.

4.4.2 Packet Filtering Module

Users can configure to capture only types of packets that they expect like packets from certain destination/source or packets that match the protocols like TCP, UDP, ARP and ICMP. Operator ">=", ">", "==", "!=", etc will be used in the filter module.

4.4.3 Packet Analyzing Module

Each packet captured will be analyzed to get the value in the header field like Port number, length, IP, checksums. Different type of packet have different header, for example a TCP packet has TCP header while UDP packet has UDP header with different fields. Thus there will be different ways to analyze the packets.

4.4.4 Statistic Generation Module

This module will analyze all the packets and generates the IP traffic statistic which shows the total number of bytes and packets sent by one station to another station. This will enable user to see how many packets and byte separately generated by both stations in a single connection. This module will generate the statistic of the protocols in percentage in a graph. Another more important statistic to generate is the top ten talker of the network. The most active host is determined by the total highest bytes sent to and received from another hosts.

4.4.5 Result Displaying Module

This module display the packets captured in three different ways. First shows the packets in a list view with source/destination IP address, protocols etc. The new the packet captured will be added to the list. The second sub module is the display of the packet when in a tree-view when the user selects the packet in the list view. The tree-view will display the packet in a hierarchy first by the root Ethernet header and its nodes like source Mac address, followed by the root IP header and finally TCP header depends on the type of packet. The other module is the display of the raw packet in the rich-text box when the packet in list-view is selected. A click on the certain root or child in the tree will highlight the match portion of packet in the rich-text box.

4.4.5 Sniffer Detection Module

This module requires the user to input the IP address of the suspected host and determine whether the host is running a sniffer program by using the ping method. In this method the system will send the ICMP request packet to the intended host and if the host reply with an ICMP reply packet, then the host is running in promiscuous mode.

4.5 Non-Functional Requirements

Non-functional specifications are the constraints on services and functions which a system must operate and the standards which must be met by the delivered system.

4.5.1 User-Friendliness

User interfaces design creates an effective communication medium between a human and a program. A program must be easy to use, understandable in a way that user satisfy of its comfortable. The three rules in applying the-user friendliness are:

- Place the user in control
  - This will define interaction modes in a way that does not force a user into unnecessary or undesired actions.
- Reduce the user's memory load
  - One of the principles that enable an interface to reduce the user's memory load is by reducing demand on short-term memory. The interface should be designed to reduce the requirements to remember past actions and results.
- Make the interface consistent
  - The interface design should apply to consistent fashion where all visual information must be organized according to a design standard that is maintained throughout all screen displays.

4.5.2 Correctness

Correctness is the degree to which the software performs its required function. A program must be able to process the input data correctly according to its function and outputs the result correctly. To ensure this application quality, lots of testing and trial-and-errors will be carried out.

4.5.3 Reliability

Reliability is the extent to which a program can be expected to perform its intended function with required precision. A reliable program will enable the function executed smoothly including ability in exception handling.

4.5.4 Efficiency

Efficiency is understood as the ability of a process procedure to be called or accessed unlimitedly to produce similar performance outcomes at an acceptable or credible speed. Efficiency is measured base on response time performance, page generation speed and graphics generation speed.

4.5.5 Maintainability

System maintenance accounts would require more effort if the system is not designed according to good programming practices. Maintainability is the ease with which a program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change in requirements.

4.6 Chosen Language, Tool and Platform

4.6.1 Language and Tool

Among the programming languages C, C++, Java and Visual Basic, C# is chosen as a language to develop the project. C# is a language that enable programmer migrate easily to .NET platform as the language has roots in C, C++ and Java with adapting the best features of each plus its own new features. By combining the C# Forms library used

to write the graphical code with the C# Socket library used to write the networking code, network applications could simply created. With C# network classes, what used to take a day to write often only takes an hour or less.

C# programs are created using an Integrated Development Environment (IDE) in .NET platform which enable programmer to create, run, test and debug more conveniently thereby reduce times compare to works without IDE. Besides the process of creating an application using an IDE is referred as Rapid Application Development (RAD).

4.6.2 Platform

Windows XP is chosen as the OS I developing the project for several reasons:

- It is the more popular used and latest OS.
- It provides better integration of Windows 9X and Window NT.
- Stable and Intelligent than Windows 9x and Windows 2K or ME.
- Windows XP is somewhat faster than Windows 2000, assuming you have a fast processor and tons of memory.
- Manufacturers of existing hardware and software products are more likely to add Windows XP compatibility now than Windows 2000 compatibility.

Chapter 5

System Design

Chapter 5 – System Design

After the requirements for the system had been identified, the process to arrange the structure of the system in details is followed through the system design. Here the entire requirement is translate into system characteristics. System design is an outline that contains the design of user interface, database and details of the functionality that shows how the system should work before it is implemented.

5.1 Structure Chart

The objective of the structure chart is to show all the relation of the modules within the program. It identifies the activities that structure the program and how it is organized. In structure chart we can figure out the major functions that form the initial component can be split into detailed sub-components. Among the independent modules we can describe the interaction between them.

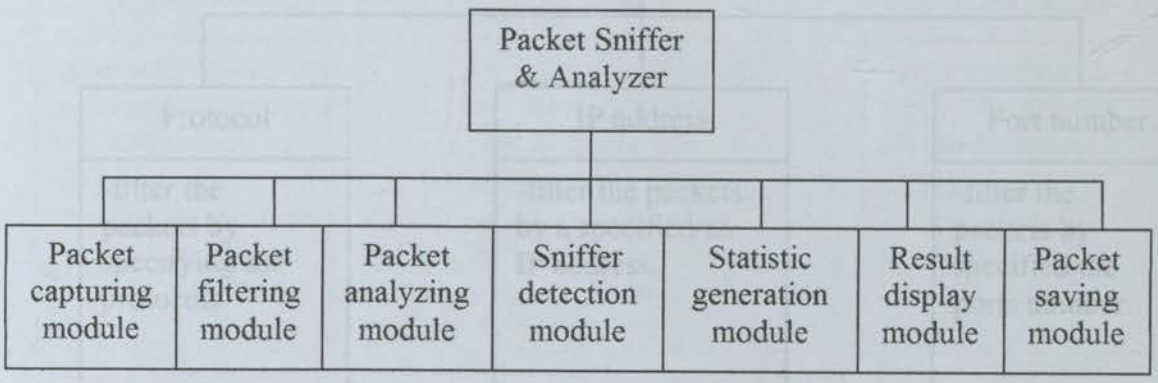


Figure 5.1.1: Structure chart for Packet Sniffer and Analyzer

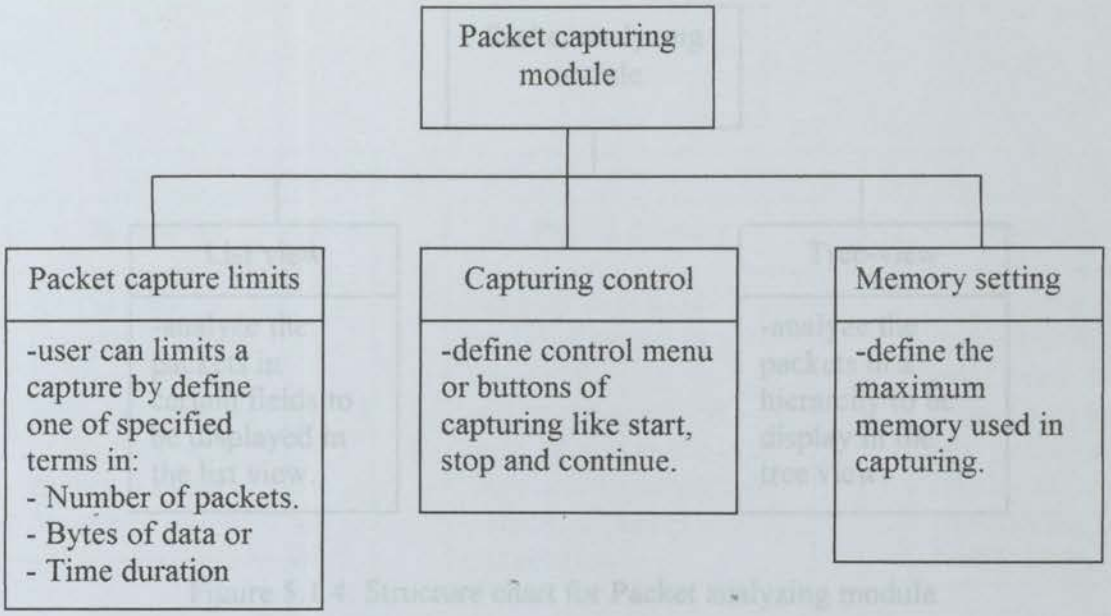


Figure 5.1.2: Structure chart for Packet capturing module

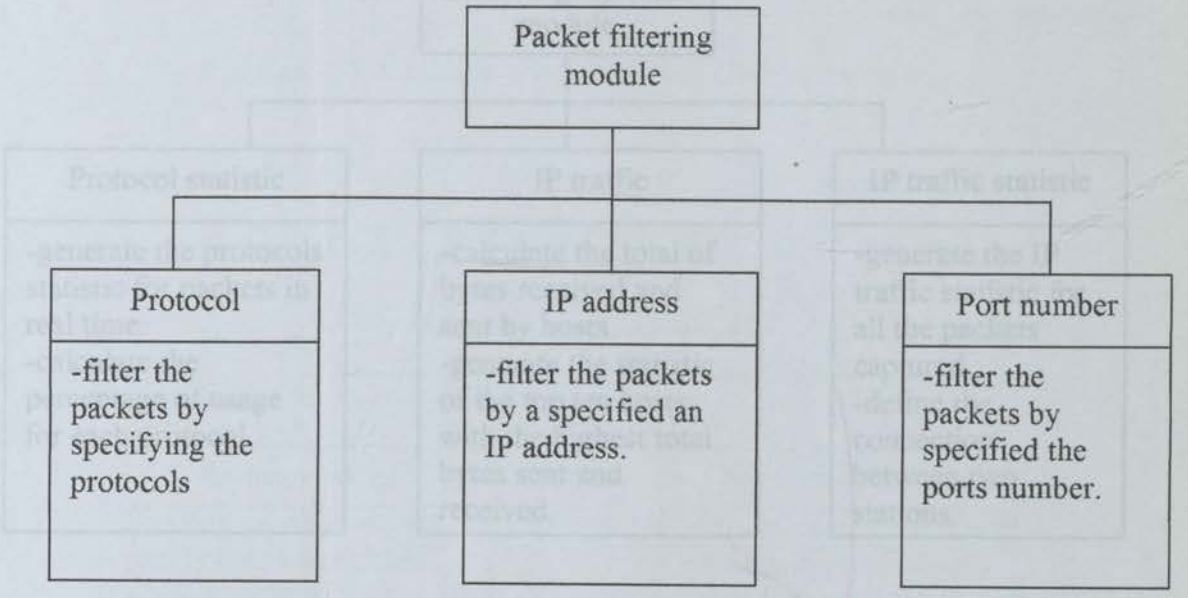


Figure 5.1.3: Structure chart for Packet filtering module

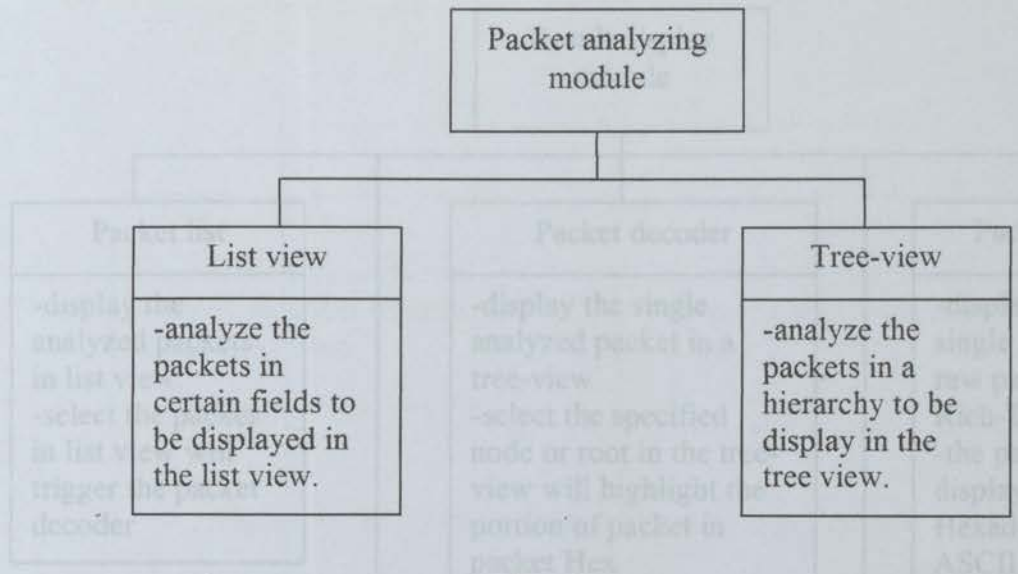


Figure 5.1.4: Structure chart for Packet analyzing module

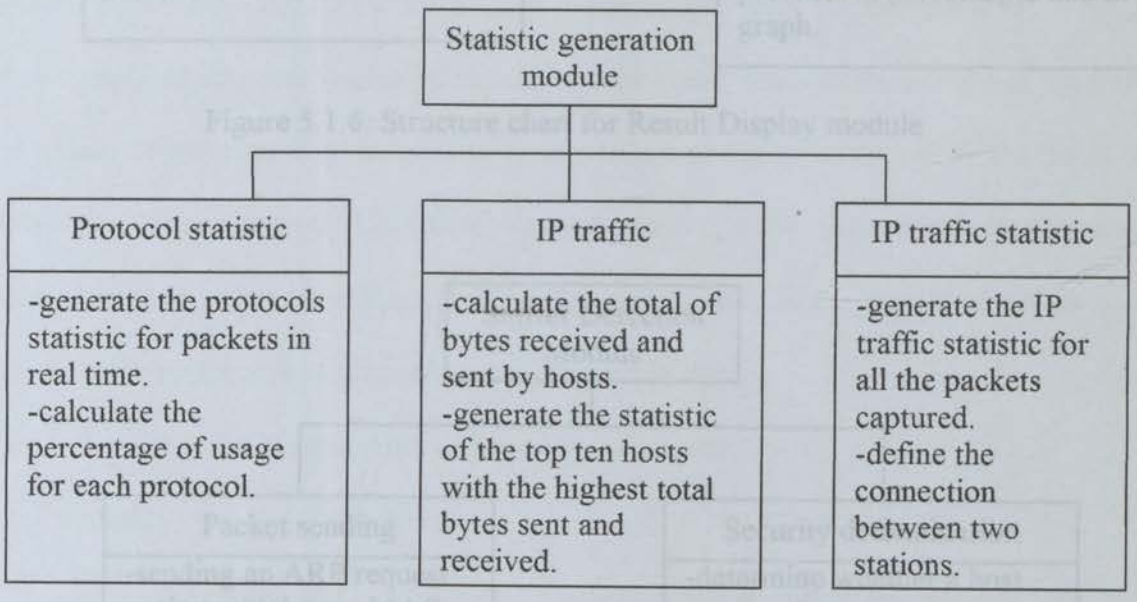


Figure 5.1.5: Structure chart for Statistic generation module

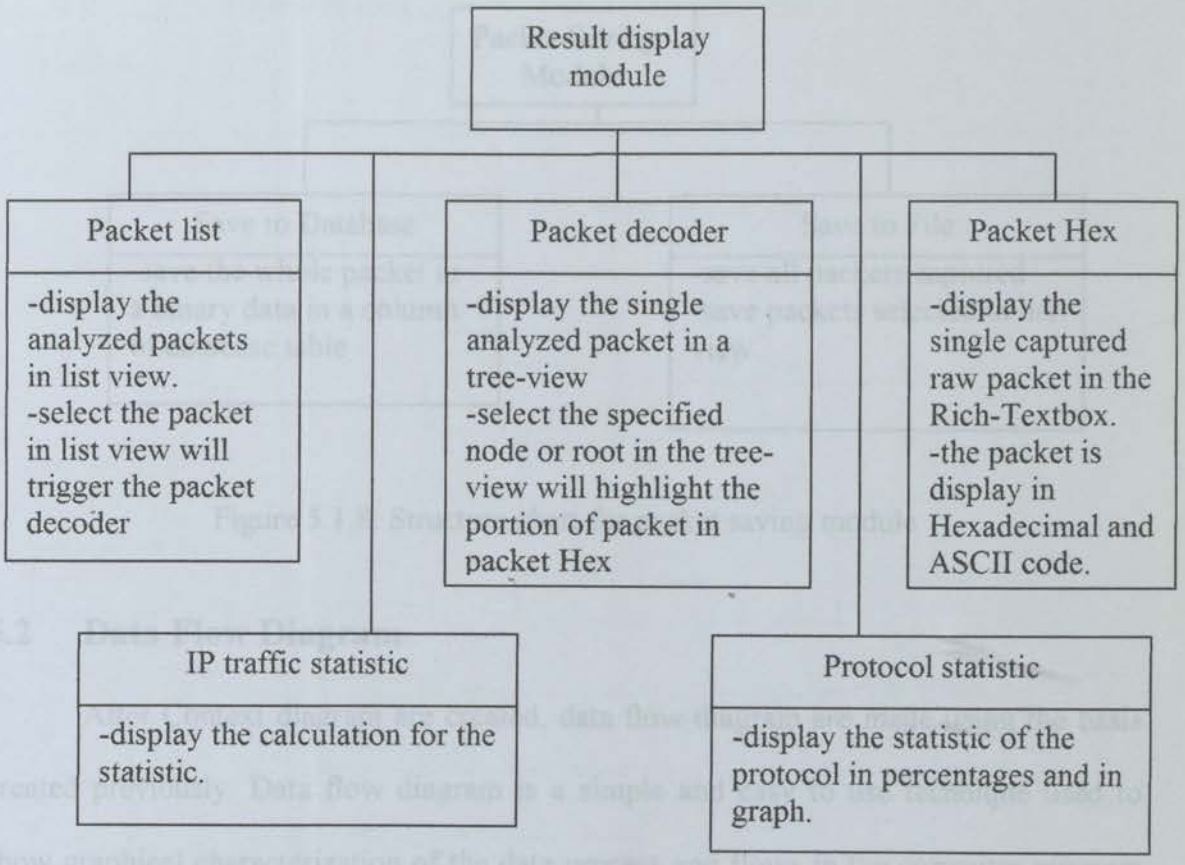


Figure 5.1.6: Structure chart for Result Display module

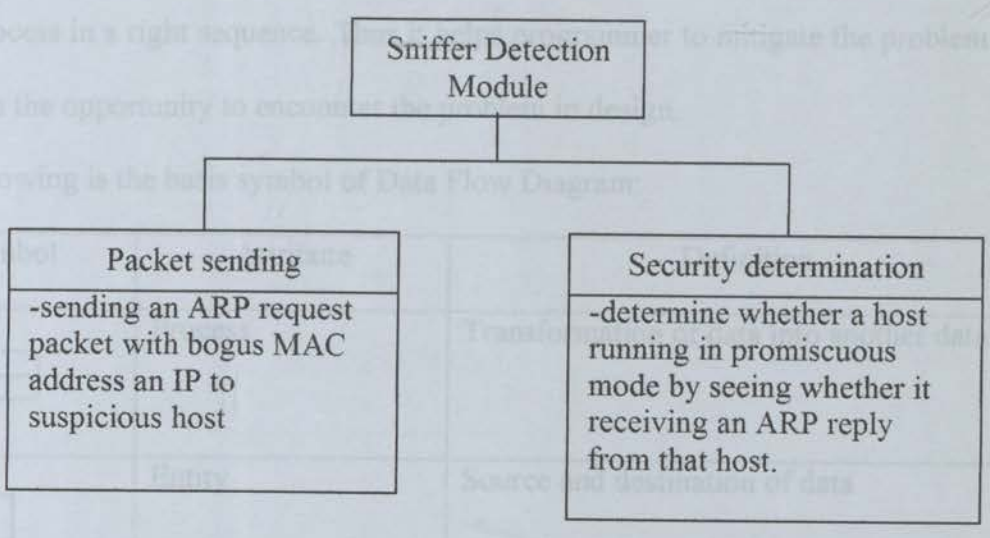


Figure 5.1.7: Structure chart for sniffer detection module

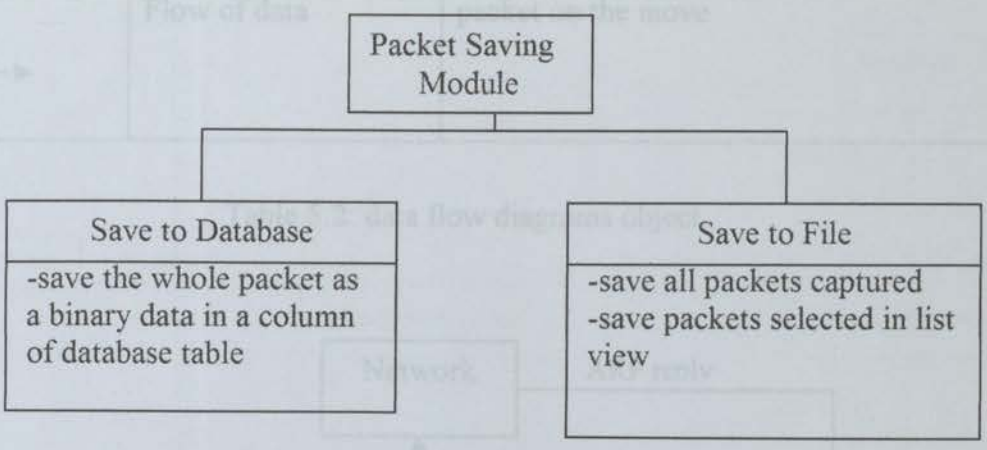


Figure 5.1.8: Structure chart for packet saving module

5.2 Data Flow Diagram

After Context diagram are created, data flow diagram are made using the basis created previously. Data flow diagram is a simple and easy to use technique used to show graphical characterization of the data process and flows in the computer program or system. It gives an overview of the system inputs and outputs as well as the flows of data through each process. Data flow diagram represents the flows of the data through each process in a right sequence. Thus it helps programmer to mitigate the problem as it provides the opportunity to encounter the problem in design.

The following is the basis symbol of Data Flow Diagram:

Symbol	Attribute	Definition
	Process	Transformation of data into another data
	Entity	Source and destination of data

→	Flow of data	packet on the move

Table 5.2: data flow diagrams object

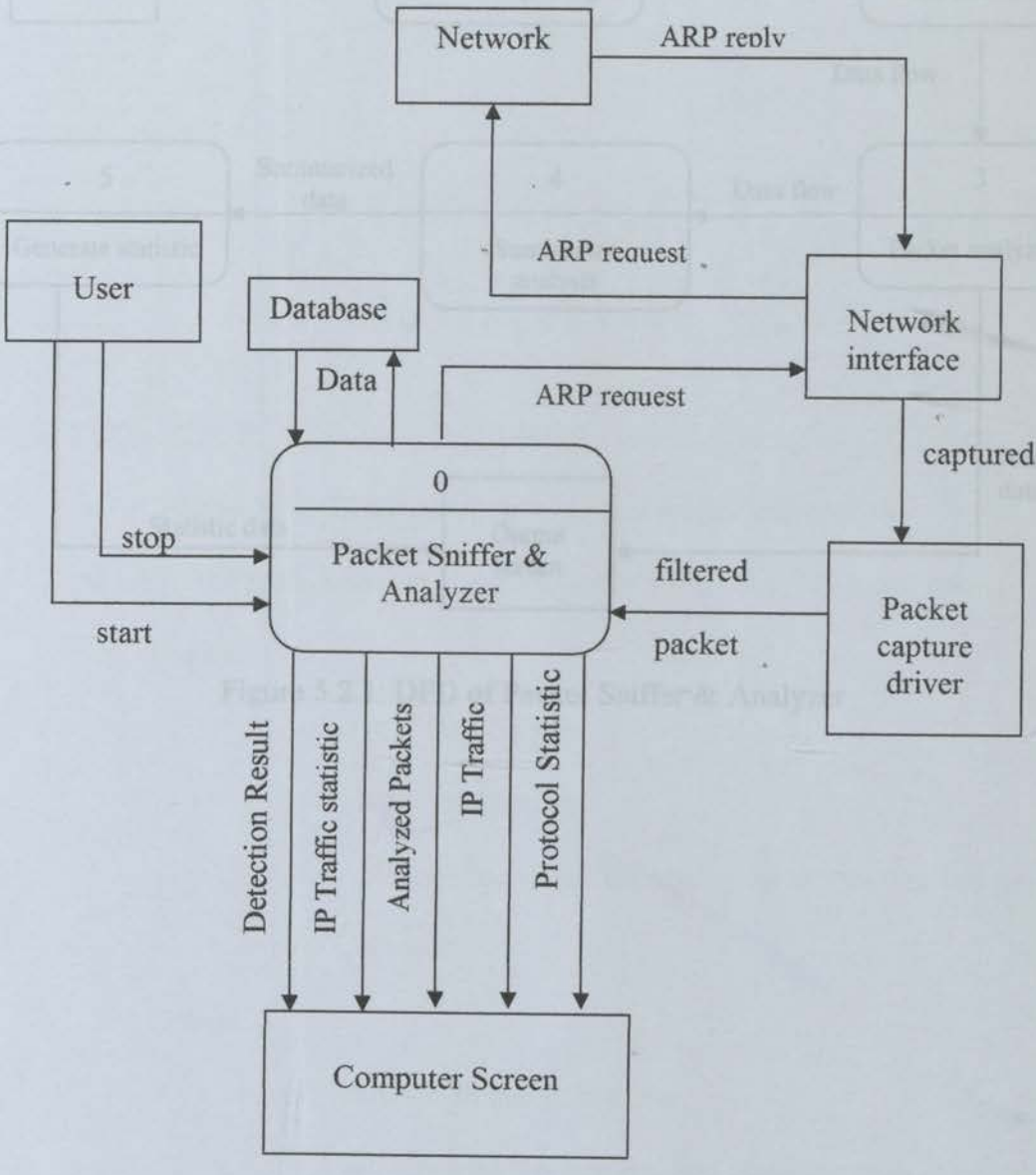


Figure 5.2.1: Diagram of Packet Sniffer & Analyzer

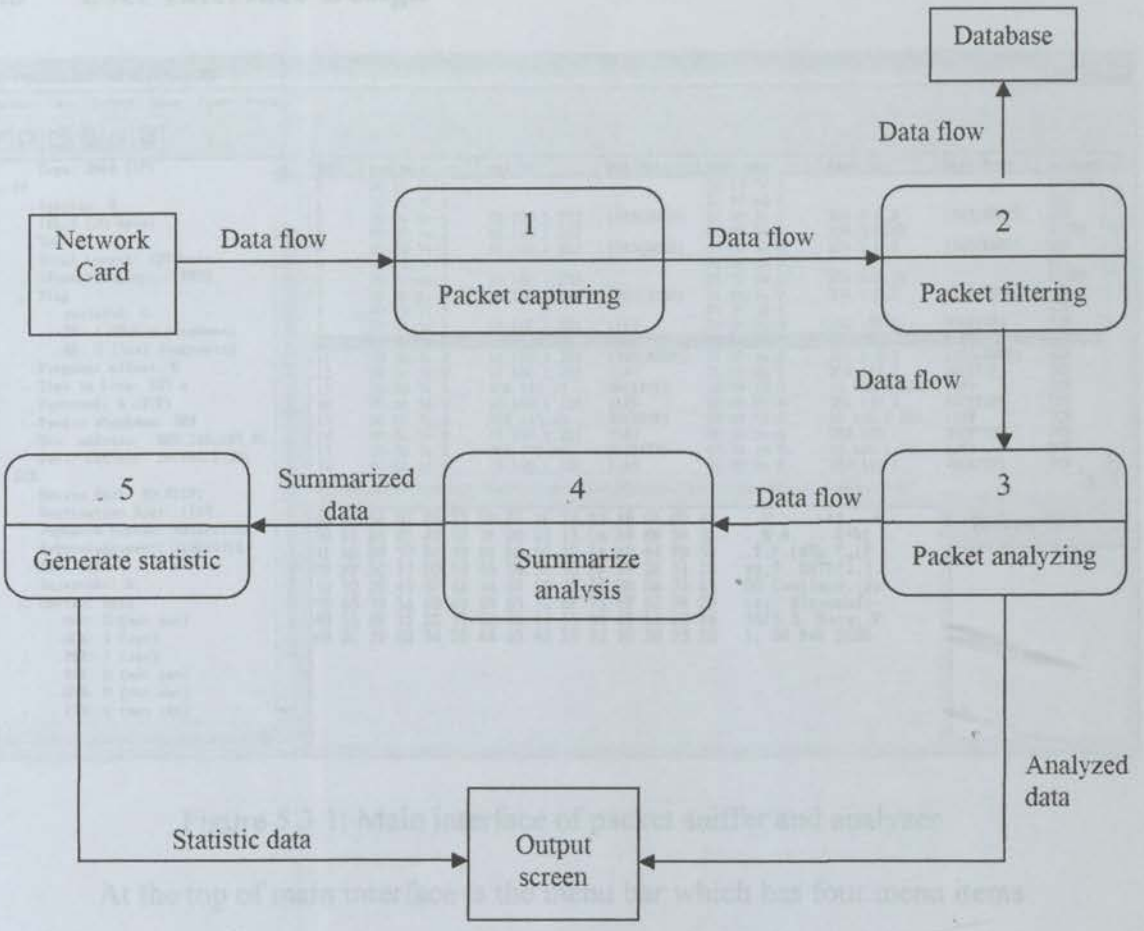


Figure 5.2.1: DFD of Packet Sniffer & Analyzer

5.3 User Interface Design

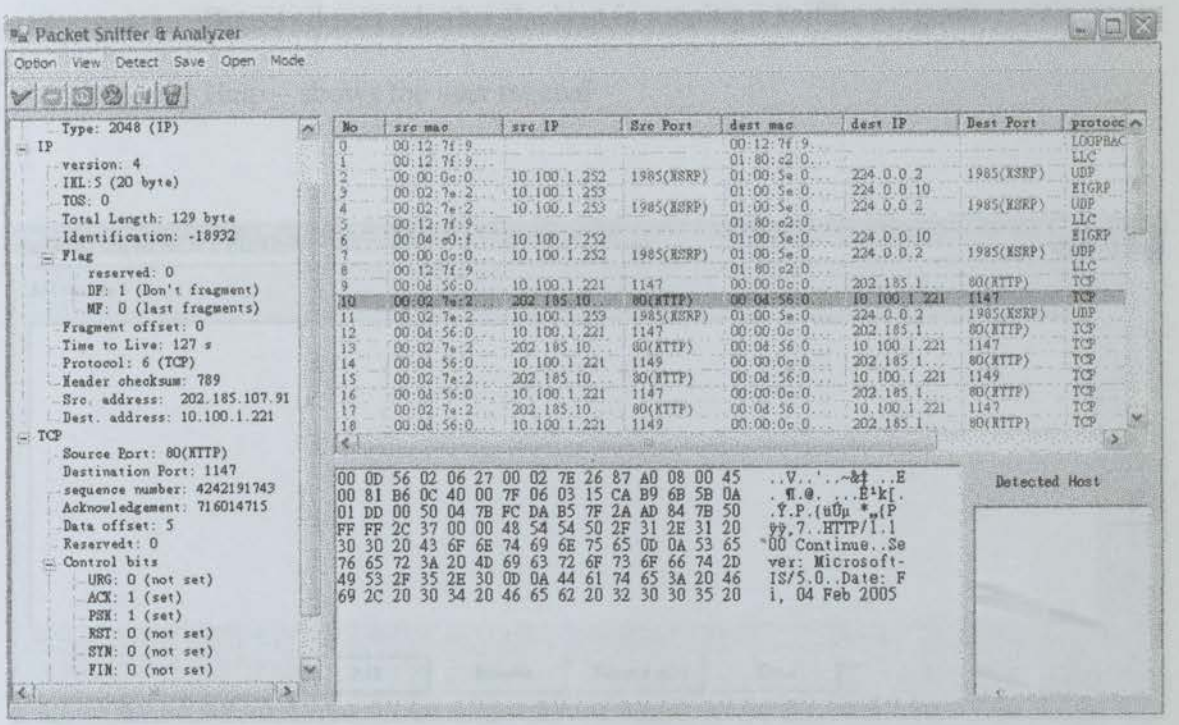


Figure 5.3.1: Main interface of packet sniffer and analyzer

At the top of main interface is the menu bar which has four menu items.

- Capture
    - *start*- start capture
    - *stop*- stop capture
    - *continue*- continue capture without clearing the previous captured
- Option
    - *filter*- load the filter menu
    - *setting*- load the memory and capture setting menu
- View
    - *IP statistic*- view the IP statistic

- *Protocol Statistic*- view the Protocol statistic bar chart

- Detect- detect whether the host is running a sniffer program.
- Help – shows the user manual

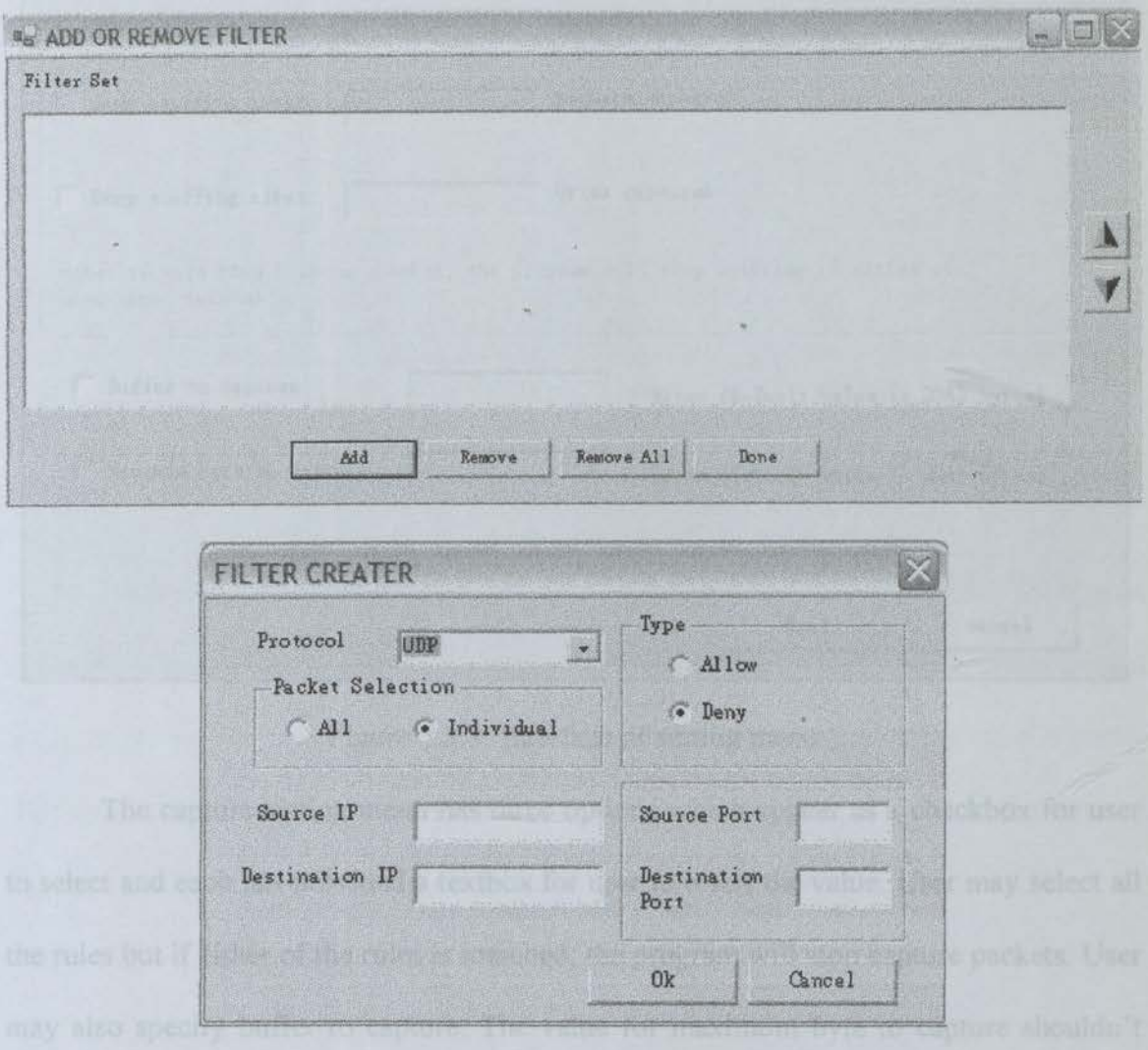


Figure 5.3.2: Interface of packet filter menu

The filter interface enable users to select protocols with option to individual or all. If the individual option is selected, users can further insert the IP address and port number as a filter statement. After user click ok, a filter item is created and it is added to the filter set.

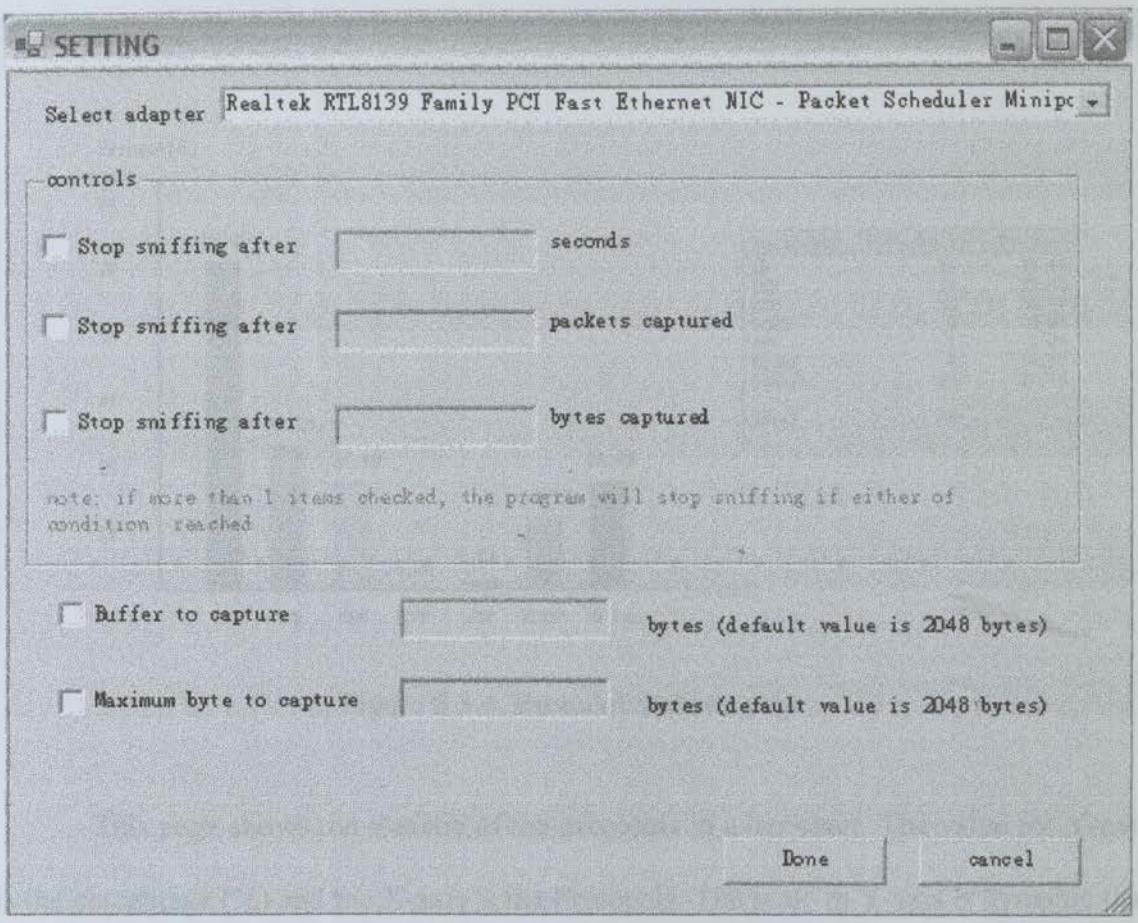


Figure 5.3.3: Interface of setting menu

The capture setting menu has three options which appear as a checkbox for user to select and each has provided a textbox for user to insert the value. User may select all the rules but if either of the rules is matched, the program will stop capture packets. User may also specify buffer to capture. The value for maximum byte to capture shouldn't exceed the value of buffer to capture.

When the done button is clicked, it will bind to the adapter assumed that the adapter is selected to capture. If user click the cancel button, it will not bind to the adapter but it still can be use to open file but if user want to start capture this interface will appear again.

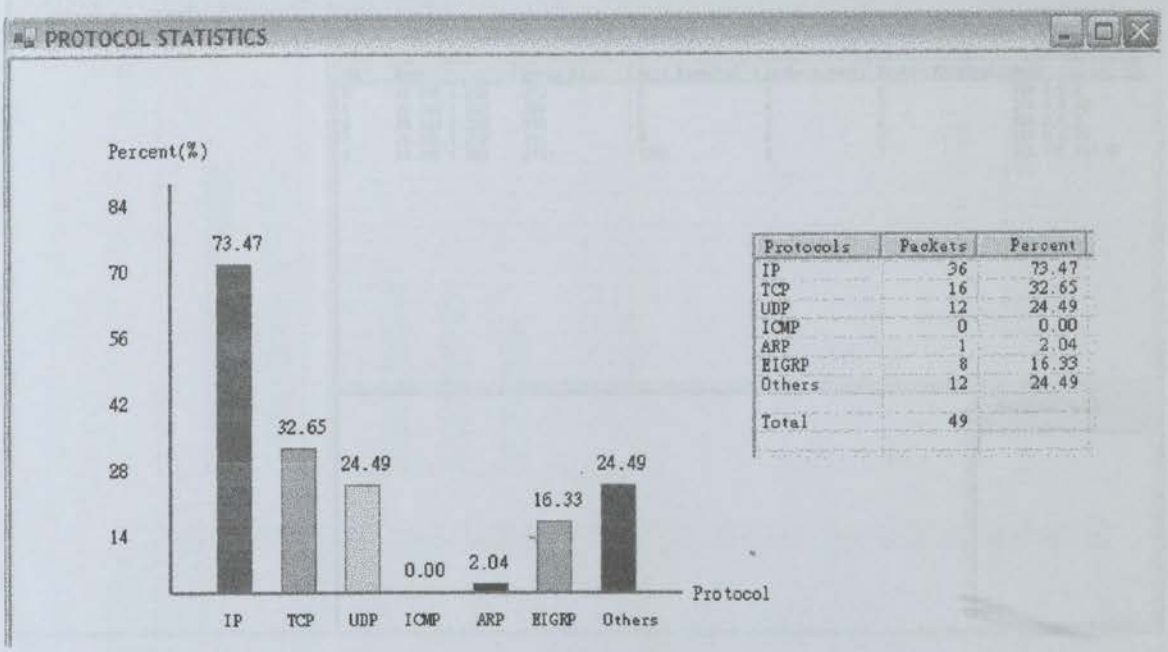


Figure 5.3.4: Protocol Statistic page

This page shows the statistic of the protocols in a bar chart. The value for Y-axis is the percentage (%) and the X-axis is the Protocols. The scale in Y-axis is dynamic that accordingly to the maximum value of the statistic but the label on the X-axis is static. The value of each protocol in percentage will be labeled at the top of the bar respectively for ease of view.

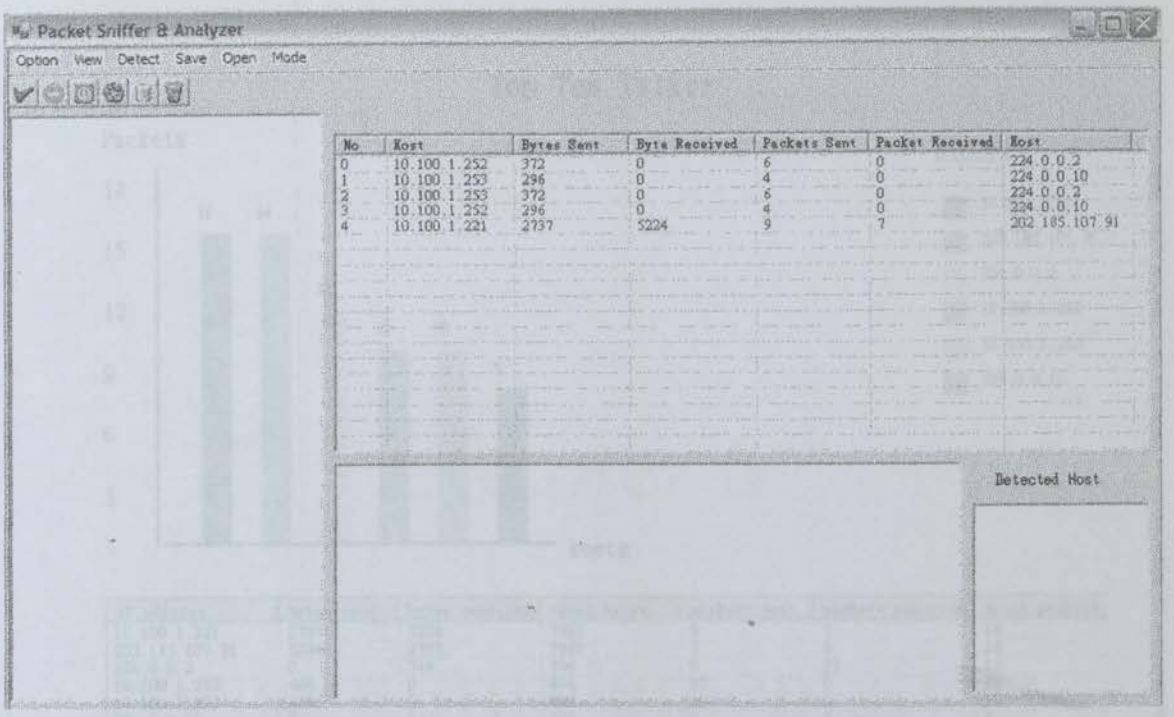
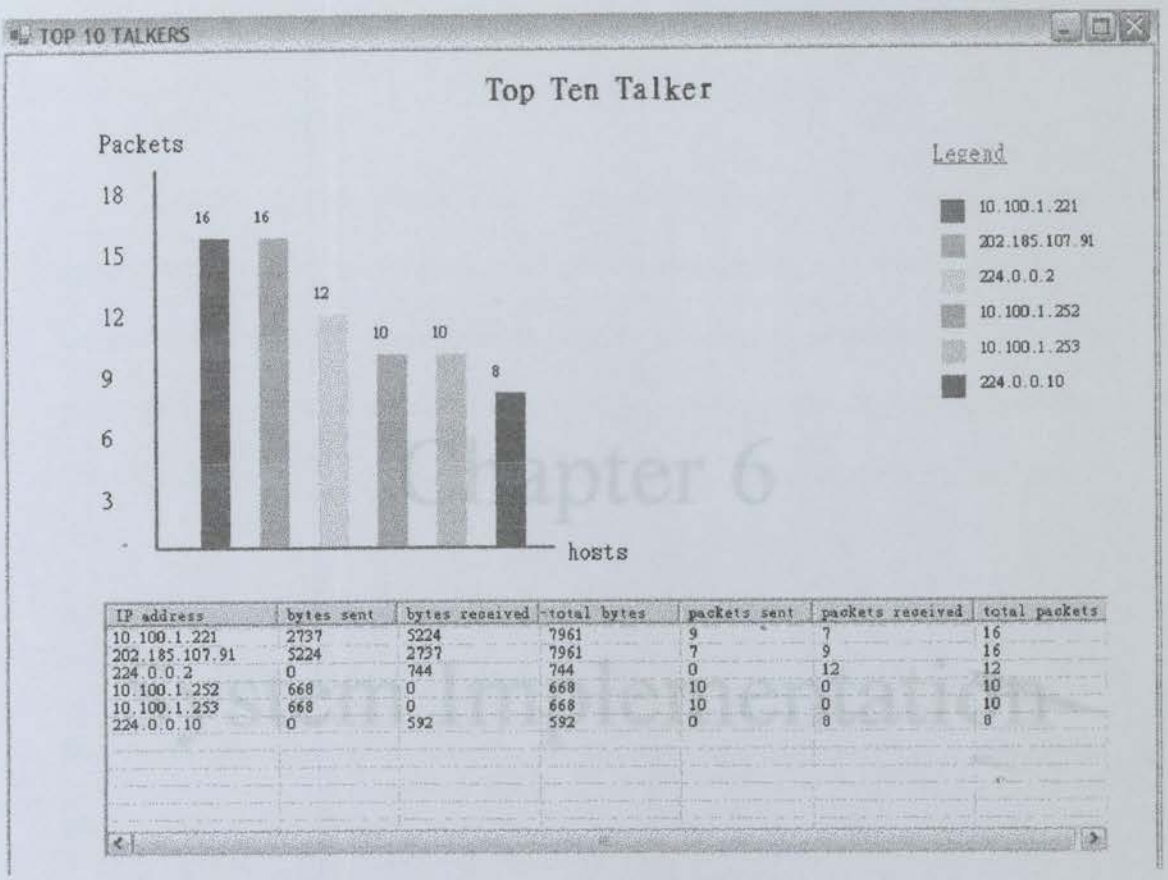


Figure 5.3.5: Connection view page

User can switch to connection view by clicking key's' if the control is on Listview or click at the menu item. The main Listview will then become invisible and the connection view will be visible. The IP Traffic Statistic page display the information of packets (IP address, number of packets sent and total bytes sent) between two stations. This summarized all the connection that established between two hosts. Each connection is displayed in one row of the list view. This page can be displayed in real time.



IP address	bytes sent	bytes received	total bytes	packets sent	packets received	total packets
10.100.1.221	2737	5224	7961	9	7	16
202.185.107.91	5224	2737	7961	7	9	16
224.0.0.2	0	744	744	0	12	12
10.100.1.252	668	0	668	10	0	10
10.100.1.253	668	0	668	10	0	10
224.0.0.10	0	592	592	0	8	8

Figure 5.3.6: IP Traffic display page

Finally, the IP Traffic page shows the top ten of the most active host in the network in a bar graph. The more detailed of the value like packets receive by, sent to and total packets is shown in the table below.

Chapter 6

System Implementation

Chapter 6 – System Implementation

6.1 Introduction

System implementation in software development is a process to convert system requirements into program codes. The implementation of the program can be separated into modules. Some modules can be developed independently but in this program the core modules are packet capturing modules and displaying modules. Once the capturing module the others modules can be implemented and tested easily.

6.1.1 Implementation of Packet Capturing Module [18]

To enable the ability of packet receiving, two features must be fulfilled. First, the program must be able to read the raw packets and secondly it able to disable the packets filter in the network interface device so that it will receive all packets. The Socket API can use for these purposes to asynchronously receive all packets. But there is a limitation where only layer 3 packets will be captured as the API is built based on protocols above layer 2.

Since the limitation of Socket API, I choose using the NDIS driver in my program to capture and send raw packets. The beauty of this driver is that the ability to work in environment without the TCP/IP protocols. This feature enable the program runs as sniffer that without having IP configuration such as IP address, network domain and subnet mask.

Before we can communicate with the driver, we must establish a handle to it by using the *CreateFile* API. Then we can query for any active adapters available and select an adapter to bind with the handle. The two processes are done through the *DeviceIoControl* API method. The binding of the adapter will let us access the NDIS driver on the adapter we selected.

After all, we can handle the device driver like a file where we can write to and read from a file. The receiving and sending of packets in this application is done through the ReadFile API and WriteFile API respectively. The Windows Driver Development Kits (DDKs) provides the NDIS driver but in manners that only send packets with own source address and receive packets that destined for our own. Therefore the following highlighted code had been added to support promiscuous mode.

```
#define NUIOO_PACKET_FILTER   (NDIS_PACKET_TYPE_DIRECTED|
                               NDIS_PACKET_TYPE_MULTICAST|
                               NDIS_PACKET_TYPE_BROADCAST|
                               NDIS_PACKET_TYPE_PROMISCUOUS)
```

The following code had been comment out to support sending packets with any source MAC address.

```
if((pIrp->RequestorMode == UserMode)&&
    !NPROT_MEM_CMP(pEthHeader->SrcAddr,
    pOpenContext->CurrentAddress,  NPROT_MAC_ADDR_LEN))
{
    DEBUGP(DL_WARN,("Write:Failing with invalid Source address"));
    NtStatus = STATUS_INVALID_PARAMETER;
    break;
}
```

6.1.2 Implementation of Packet Filtering Module

Whenever a frame is read from the driver it will invoked an event through the delegate so that the frame in byte[] is passed to a check filter function to determine whether the frame is to be keep or dropped. The filter is made in sort of statement where each statement is an object of a class called *FilterItem* that keep in the ArrayList named *FilterList* which is an attribute of class *FilterManager*. The class *FilterItem* is composed of attributes like protocols, source and destination IP address and port number etc. The frame is checked with all the filter items made by user by

comparing the items' attribute with the frame's attribute. The filtering function will return a Boolean value. If true returned, the frame will keep for further analyzing.

6.1.3 Implementation of Displaying Module

I declare a public struct named "Collections" where it contains a static ArrayList called "Captured" and others public members are fields to be displayed in the ListView. Note that the ArrayList is declared static so that only one version of that array is created and it is only belongs to the struct instead of the object of the struct. Whenever a packet after filtered and if found it is to be keep, a new object of struct Collections will be created. Then the whole datagram will be added to the Captured ArrayList. The program will continue to analyze the packet and assign the value to members of struct. I also declare a static HashTable named "PacketTable" to store the object of struct "Collections" with the packet Id as the key. After all, the object of struct will be passing to "addListView" function to display it in the ListView.

Whenever a row in the ListView is selected, an event will be fired to display the more details value of that packet. I retrieve the Id of the packet from the column "no." in the ListView. The Id will be the index I used to retrieve captured packet in "Captured" ArrayList. Since some of the information already analyzed stored in the struct, I prefer to reuse it rather than analyzed it once again. So I get the reference of the object stuct "Collections" from the "PacketTable" by using the index and pass it together with the packet to the "EthernetParser" function. This function will display the packet's fields in tree view. the "writeToRTB" is another function use to display the packet in Hexadecimal and ASCII code.

6.1.4 Implementation of Packet Analyzing Module

The frame captured is keep in an array of type byte (byte[ ]), therefore the size of the frame in bytes is determined by the size of the array. Each element of the array is an 8-bit integer where the maximum value is 255 in decimal or FF in hexadecimal. Each element in the array can be converted in ASCII code by performing casting with data type char. To analyze the frame for more readable format, a function called ToUInt is used to convert to unsign integer.

```
public static uint ToUInt(byte[] datagram, int offset, int length)
{
      uint total = 0;
      int byte_index;
      int bit_offset;
      int bit;
      byte b;

      for ( int i = 0; i < length; i++ )
      {
         bit_offset = (offset+i) % 8;
         byte_index = (offset+i-bit_offset)/8;
         b = datagram[byte_index];
         bit = (int)(b >> (7 - bit_offset));
         bit = bit & 0x0001;

         if ( bit > 0 )
         total += (uint)Math.Pow(2,length-i-1);
      }
      return total;
}

public static int ToInt(byte[] datagram, int offset, int length )
{
      return (int)ToUInt(datagram,offset,length);
}
```

The function receives three arguments. The argument offset represents the index in bits and the argument length represents the length of bits to analyze from the offset. To handle variable kind of return type we can simply add the casting to the *ToUInt* function.

```
public static String GetIPString(uint addr)
{
      uint a = addr >> 24;
      uint b = (addr >> 16) & 0xFF;
      uint c = (addr >> 8) & 0xFF;
```

```
        uint d = addr & 0xFF;
        String format = "{0}.{1}.{2}.{3}";

        return String.Format(format,a,b,c,d);
}
public static string GetMACAddress(byte[] PacketData, int Index)
{
        string Tmp = "";
        int i = 0;
        for(i=0;i<5;i++)
        {
            Tmp += PacketData[Index++].ToString("x02") + ":";
        }

        Tmp += PacketData[Index++].ToString("x02");
        return Tmp;
}
```

The codes above shows functions to obtain the IP address and MAC address respectively. The '<<' is right shift operator that shifts left operand right by a number of bits on the right whereas the '&' operator performs the bitwise AND operation of its operands.

5.1.5 Implementation of Statistic Generation Module

I declare a delegate named "analyzing" which will invoke an event in the "CheckFilter" function to updating the IP statistic which will be used to generate connection statistics Top-10 talker statistics. The following is the static ArrayList I declared to store the information for connection statistic.

source:	source IP address of an IP packet
destination:	target IP address of an IP packet
counter:	number of packets sent or received countered for that connection
senderSend:	number of bytes sent by the source in a connection
senderRec:	number of bytes received by the source in a connection
sent:	number of packets sent by the source in a connection
receive:	number of packets received by the source in a connection

An event invoked will pass the source and destination address and byte of the packets to the "generateHosts" function. The function whether the packet is found in existing connection else a new connection will created for it. For example, if I receive a packet where the source is 'A' and destination is 'B' with size of byte 'X'. The first element of source will be filled with 'A' and destination filled with 'B'. Both counter and sent will be filled with value 1. The first element of senderSend, senderRec and receive ArrayList will be filled with 'X', 0 and 0 respectively. If a second packet arrive with source 'B', destination 'A' and byte 'Y', no new connection will be created but the counter and receive will increment by 1 and the senderRec will be added with 'Y'.

To generate the talkers, again I provide the following static ArrayList:

`normalized`:	talker's IP address
`amount`:	number of packets sent or received by the talker
`sentByte`:	number of bytes sent by the talker
`recByte`:	number of bytes received by the talker
`sentTo`:	number of packet sent by the talker
`receivedBy`:	number of packet received by the talker

The talkers' statistic is done through the "calculateHost" function. First, the function will add the hosts in first connection to the first talker and second talker. Then the program traverses the other connections, checking the source and destination host with each the hosts in "normalized" ArrayList. If a match is found, the value in talker ArrayList will be updated; else a new element of talker will be created.

After traversing the connections, a non-duplicate of talkers is generated. Each element of the talker will be sorted descending based on total number of packets sent and received before pass to IpStatistic class to generate graph.

5.1.6 Implementation of Packets Storing Module

The program provides two options of storing packets captured; saving into files and saving into Microsoft SQL server. I assume that there is a local database named Sniffer with table Packet and Session. Table Packet has column fields "PacId", "Datagram" and "SessionName". Before user can save the packets, a session name of the capture must be established first so that we able to load and delete the packets based on session name.

```
Private void saveToDatabase(byte[] datagram)
{
   SqlConnection conn = null;
   SqlCommand cmd =null;
   SqlParameter Id =null;
   SqlParameter FileName =null;
   SqlParameter data =null;

   try
      {
       conn = new SqlConnection(ConnectionString());
// We assume there is a stored procedure called Upload
       cmd = new SqlCommand("Upload",conn);
       cmd.CommandType = System.Data.CommandType.StoredProcedure;

       Id = new SqlParameter("@PacId",System.Data.SqlDbType.BigInt,8);
       Id.Direction = ParameterDirection.Output;
       data = new SqlParameter("@Datagram",SqlDbType.Image);
       data.Value = datagram;
       FileName = new SqlParameter("@SessionName",SqlDbType.Char,20);
       FileName.Value = sessionName;

       cmd.Parameters.Add(data);
       cmd.Parameters.Add(FileName);
       cmd.Parameters.Add(Id);

       conn.Open();
       cmd.ExecuteNonQuery();
       cmd.Dispose();
       datagram=null;
      }
      catch(Exception e)
      {
        MessageBox.Show(e.Message + " - " + e.StackTrace);
      }
      finally
      {
        conn.Close();
        conn.Dispose();
      }
}
```

The previous codes show the function of saving packets using stored procedure. It is important to close the connection after execution. Use the finally with try and catch exception handling method so that the connection will always close at the end even if an exception occurred. If we store the data into local SQL server and using windows authentication to make connection with SQL server, then the connection string we declare can be as simple as follow:

```
public static string ConnectionString()
{
   return "server=.; database= Sniffer; Integrated Security=SSPI";
}
```

I provided two options of saving packets into file using SaveFileDialog method as well; saving all packets into and saving selected packets in the ListView. C# [19] views each file as a sequential stream of bytes. When a file is opened, an object will be created to associate it with a stream. FileStream is one of class inherited from class Stream under namespace System.IO that provide definition for both input from and output to a file. C# provides class BinaryFormatter which is used in conjunction with a Stream object to perform input and output of objects. Serialization involves converting an object into a format that can be written to a file without losing any of object's data. Deserialization consists of reading this format from a file and reconstructing the original object from it. A BinaryFormatter can serialize object to, and deserialize objects from, a specified Stream.

```
if(filename==""||filename==null)
   MessageBox.Show("Invalid File
name","ERROR",MessageBoxButtons.OK,MessageBoxIcon.Error);
else
{
try
{
 output=new FileStream(filename,FileMode.OpenOrCreate,FileAccess.Write);
}
  catch(Exception)
{
    MessageBox.Show("File not
```

```
exist","error",MessageBoxButtons.OK,MessageBoxIcon.Error);
}
}
try
{
if(packetTable.Count>0)
    formatter.Serialize(output,Collections.captured);
}
catch(SerializationException)
{
  MessageBox.Show("error writing to
file","error",MessageBoxButtons.OK,MessageBoxIcon.Error);
}
```

The previous codes show part of function on saving all packets into a file. Note that I pass the static ArrayList "Captured" from struct "Collections" to be serialized by the BinaryFormatter.

To save the selected packets, we can choose the packets in ArrayList "Captured" and add it into a temporary ArrayList as shown in code below.

```
try
{
  if(packetTable.Count>0)
  {
    ListViewItem LItem;
    int index=0;
    for(int i=0;i<listView1.SelectedItems.Count;i++)
    {
      LItem = listView1.SelectedItems[i];
      index = LItem.Index;
      Collections.temp.Add(Collections.captured[index]);
    }

      formatter.Serialize(output,Collections.temp);
      Collections.temp.Clear();
   }
}
catch(SerializationException)
{
    MessageBox.Show("error writing to
file","error",MessageBoxButtons.OK,MessageBoxIcon.Error);
}
```

To open a file, the deserialization must be performed as shown in code below.

```
input=new FileStream(filename,FileMode.Open,FileAccess.Read);
Collections.captured=(ArrayList)reader.Deserialize(input);
```

6.1.7 Implementation of Sniffer Detection Module

The detection of sniffer can be done in many methods. I am using ARP request as a detection of hosts running in promiscuous mode. To form an ARP request packet, three fields are required for user to input. They are sender's MAC address, sender's IP address and target's IP address. For more accuracy of detection, it is better to put the fake MAC and IP address of the in the sender fields since it may have the possibility that any host sending an ARP reply due to ARP request generated by our own machine. Both of sender's MAC and IP address must be set to fake, if only either one fake, then it will cause conflict whenever a host is detected.

I am provided another option for user to send a single ARP request or many ARP request by getting the host address from the database.

Chapter 7 – System Testing

7.1 Introduction

The main function of testing is to detect the presence of defects in a program and to judge whether the program is usable in real application. Testing can only demonstrate the presence of errors not to show that there is no error. Therefore, a more suitable approach must be chosen to reduce the possibility of errors in a program.

Bottom-up approach is the testing method used in early implementation of project since it was clearly each modules work independently before integrated. There is no a phrase for testing but it is done through the life cycle. Each module at the lowest level of the system hierarchy is tested individually. Then, all the tested modules would be related to the next module testing. This approach is repeated until all the modules are tested successfully.

7.2 Testing Process

In general, the testing process of ELONS can be shown in the following figure. All the details will be further explained in subsequent sub-sections

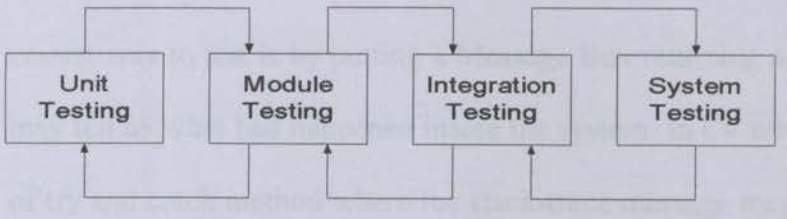


Figure 7.1 Testing Process

7.2.1 Unit Testing

- In Unit testing the individual component is tested to ensure that they function properly. They are tested independently without the interference with other system components.
- Techniques used during the process of performing unit testing are as follows:
    - **Code Review**

      Codes are reviewed line by line to discover any syntax error as well as semantic error. If errors are discovered, they are corrected immediately. In C# some syntax errors may be shown even before compilation.
    - **build of solution**

      This is faster way compared to code review techniques and it is efficient in discovering errors as the compiler will detect type of errors in a program and display the error type as well as the line number in which the error occurs.
    - **Other techniques**

      If the error occurs during the loop of a function or it is no caused by syntax error, then it will be difficult to identify the actual error. The easiest way to test is by putting a Message Box returning a value or even may tell us what had happened inside the system. In C# we can make use of try and catch method where the stack-trace message may trace for any exception occurred.

7.2.2 Module Testing

Module testing is performed without other system modules. A module consists of a collection of dependent components to perform a particular task or function. Different possible test cases are applied to the module and the test results would be verified. Unusual results will be analyzed and they would help in debugging sub-modules in order to produce the desired output. The module testing can be done by creating a new project with the defined sources and predicted results. Besides, the accuracy of the program can increase by comparing the results with that one generated by existing program.

7.2.3 Integration Test

Integration test is needed when all modules are integrated. The main focus in integration test is to navigate the interfaces repeatedly to detect any mismatch of data type or variable. Several important aspects are checked to ensure that the flow of the data is well organized and are user friendly to users.

7.2.4 System Test

The sub-systems are integrated to make the entire system. Therefore, the main purpose in system testing is to find errors that result from unanticipated interactions between sub-systems. Besides, it is used to validate whether the system meets its functional and non-functional requirement.

Finally, a performance test is performed to compare the integrated modules with the non-functional system requirements such as interoperability, flexibility and reliability.

Chapter 8

System Evaluation

Chapter 8 - System Evaluation

8.1 Introduction

Evaluation is the final phase of developing a system and an important phase before delivery the system to the end users. Evaluation was related to user environment, attitudes, information priorities and several other concerns that are to be considered carefully before effectiveness can be concluded. At all phases of the system approaches, evaluation is a process that occurs continuously, drawing on a variety of sources and information.

8.2 System Strengths

The program I developed has the basic ability to capture packets from all layers without limited to only TCP and UDP packets. Therefore it can be used to help network engineers to captures packets that involving communications between active devices such as routers. With the sniffing of EIGRP packets we able to know which device is active or down.

The program allow network engineer to save packets into a database rather that just into a file and the capturing of the packets can be stopped automatically if it exceed certain number of packets, size of packets and a certain period. I also provided option to set the buffer used to capture at the low level. The default value is 2048 bytes. But if we would like to capture only the header of the packets, we can set the buffer as low as it exceeds the header length of the packets. Therefore it will reduce the memory in used. The filtering I developed also may helps users to filter and focus on packets he likes to receive.

I developed the statistic module where users can get an overview of the packets he captured. We will be able to identify the active talkers among the hosts in a network. With the connection view, we can see clearly how many connections have been established among the hosts without checking the packets one by one.

Sniffer detection is developed not only to detect a single host but I allow users to detect a list of hosts from database and sending each of them an ARP request at a time.

8.3 System Constraints and Future Enhancements

As there are many types of packets with its own protocol and different data type, the analyzing of the packets still limited in my project which only focus only the regularly used protocols such as TCP and UDP, ARP, ICMP and EIGRP. The analyzing of packets still limited until transport layer of the OSI model. It can be further enhanced to analyze until the application layer such as HTTP and to follow the TCP stream.

The existing system can be used to sniffing for an authentication process. If there is no encryption applied, the user login's information can be seen in the text area. But it requires us to manually do trace the packets one by one. It can be enhanced to automatically search for the information and generate reports to us.

Since the sniffer has the ability to analyze IP address and hardware address of a packet, it has the potential to be used as a tool to create a database where both the address is save so that it provides reference to network engineers in checking any conflict of IP address and Mac address among the computers before a new IP address will be assigned to new devices.

Since the current program only provide saving packets into local database, it can be optionally to connect to others SQL server in the network and saving the packets into that server's database. Then the sniffer should act like a buffer where it releases the memory whenever it captures a single packet, so that the memory usage will not keep increasing. It is also possible that we develop another web based system to read the packets through the Internet. Then we will able to monitoring a network through the Internet access.

Besides, the program can be enhanced to be more intelligent where it allow user to have more options or customizations. For example, it allowed user to specify a specify range of time to sniff and it allows user to keep on repeating detecting sniffer after a certain periods as what user being set.

REFERENCE

[1] Case study 1: Available at:
http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=42170&lngWId=1

[2] Case study 2: Available at:
http://www.analogx.com/contents/download/network/pmon.htm

[3] Case study 3: Available at:
http://www.network-monitor.com/?js=on

[4] Case study 4: Available at:
http://www.etherdetect.com/

[5] Case study 5: Available at:
http://www.gjpsoft.com/UltraNetSniffer/

[6] http://www.securityfriday.com/promiscuous_detection_01.pdf

[7] http://www.packetwatch.net/documents/ papers/snifferdetection.pdf

[8] NDIS. Available at:
http://winpcap.polito.it/docs/docs31beta3/html/index.html

[9] WinPcap. Available at:
http://winpcap.polito.it/docs/docs31beta3/html/index.html

[10] http://www.handels.gu.se/epc/archive/00003725/01/Nr_3_DS.pdf

[11] Internet Protocol. Available at:
http://www.faqs.org/rfcs/rfc791.html

[12] Transmission Control Protocol. Available at:
http://www.faqs.org/rfcs/rfc793.html

[13] User Datagram Protocol. Available at:
http://www.faqs.org/rfcs/rfc768.html

[14] Address Resolution Protocol. Available at:
http://www.faqs.org/rfcs/rfc826.html

[15] Internet Control Message Protocol. Available at:
http://www.faqs.org/rfcs/rfc792.html

http://www.informit.com/articles/article.asp?p=26557&seqNum=4

[16] ASCII code. Available at:
http://www.cplusplus.com/doc/papers/ascii.html

[17] Schach, Stephen R. (2003) *Object-Oriented and classical Software Engineering* –6th edition. McGraw-Hill.

[18] Raw Ethernet Packet Sending, Miahrugger. Available at:
http://www.codeproject.com/csharp/SendRawPacket.asp

[19] Deitel, Deitel, Listfield, Neito, Yaeger, Zlatkina. (2002) C# How To Program. Prentice Hall.

Source code reference

1. Raw Ethernet Packet Sending, Miahrugger. Available at:
http://www.codeproject.com/csharp/SendRawPacket.asp

2. Packet Capture and Analayzer, fkocak. Available at:
http://www.codeproject.com/csharp/pacanal.asp

3. Bar Chart In C#, Syed Hasan Adil, Available at:
http://www.csharphelp.com/archives/archive254.html

4. Retrieving Hardware Identifiers in C# with WMI, Peter A. Bromberg, Available at:
http://www.eggheadcafe.com/articles/20030511.asp

5. Uploading / Downloading Pictures to / from a SQL Server, Alexandru Ghiondea, Available at:
http://www.codeproject.com/cs/database/UploadPicturesSQLServer.asp